强化学习
从AlphaGo Fan到AlphaGo Zero

刘畅
20180515

AlphaGo

Review Reinforcement Learning

RL in AlphaGo

评价体系　　仿真搜索　　经验转化

Different versions of AlphaGo

AlphaGo Lee　　AlphaGo Zero

So far...Supervised Learning

Data: (x, y), x is data, y is label

Goal: Learn a function to map x->y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Cat

Classification

So far...Unsupervised Learning

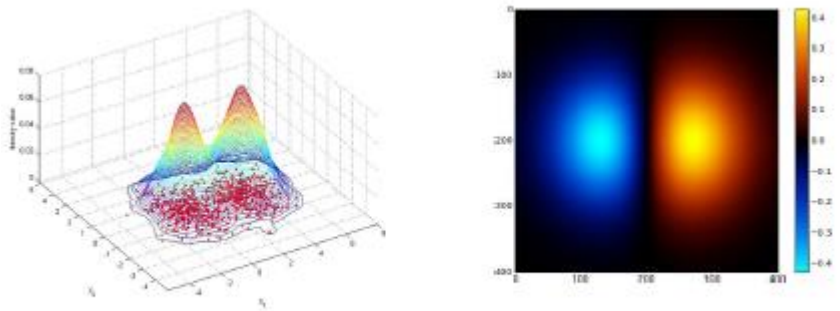
Data: x, x is data, no labels!

Goal: Learn some underlying hidden structure of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

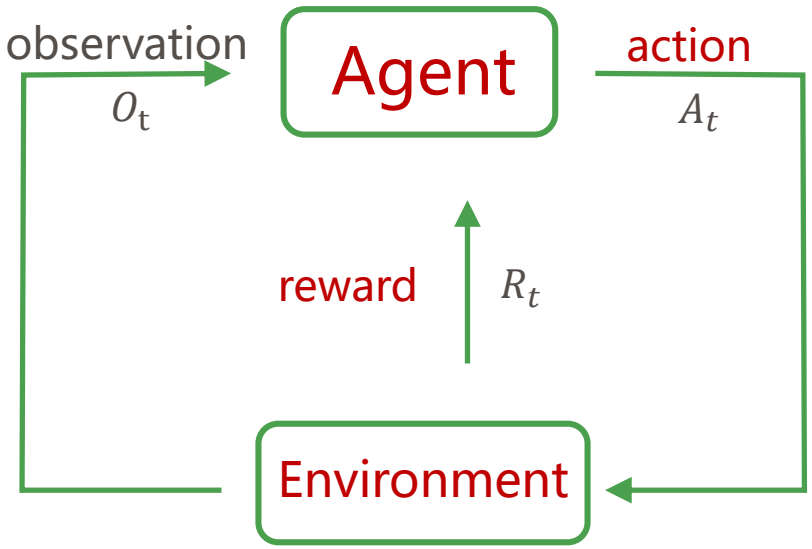
Today: Reinforcement Learning

Problems involving an **agent** interacting with an **environment**, which provides numeric **reward** signals

Goal: Learn how to take actions in order to maximize reward.


感知
(深度学习)
智 能
决策
(强化学习)

Major Components of RL



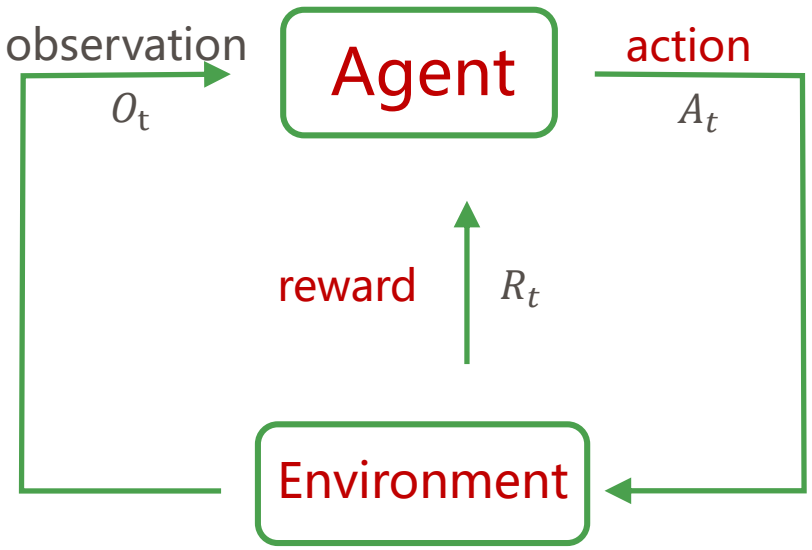
- A reward R_t is a scalar feedback signal , indicates how well agent is doing at step t. The agent’ s job is to maximize cumulative reward.
- Sequential decision making
  1. Goal: select actions to maximize total future reward.
  2. Actions may have long term consequences.
  3. Reward may be delayed.
  4. It may be better to sacrifice immediate reward to gain more long-term reward.
- The **history** is the sequence of observations, actions, rewards.

$$H_t = O_1, R_1, A_1, \ldots\ldots, A_{t-1}, O_t, R_t$$

- **State** is the information used to determine what happens next. Formally, state is a function of the history:

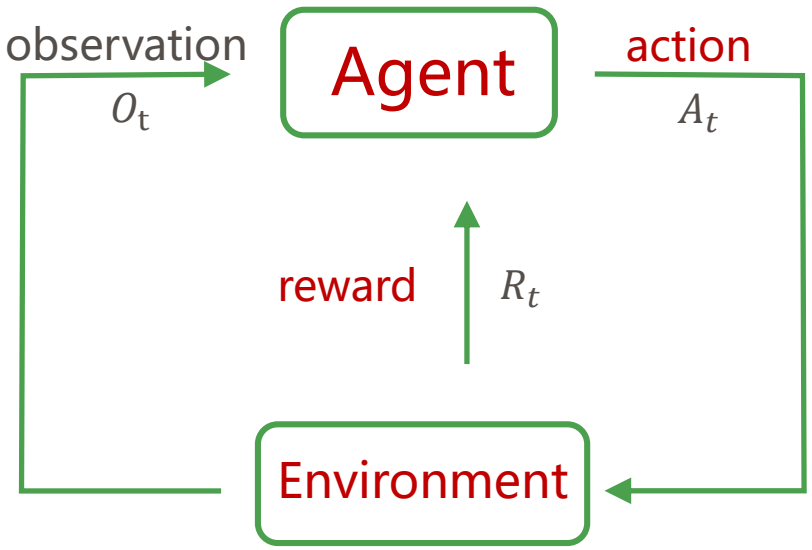
$$S_t = \mathrm{f}(H_t)$$

Major Components of RL



- A state S_t is Markov if and only if
  $P[S_{t+1}|\ S_t] = P[S_{t+1}|\ S_1, \dots, S_t]$
- Fully Observable Environments
  1. Fully observability: agent directly observes environment state. $O_t = S_t^a = S_t^e$
  2. Formally this is a Markov decision process(MDP)
- Partially Observable Environments
  1. Now agent state ≠ environment state
  2. Formally this is a partially observable Markov decision process(POMOP).

Major Components of a RL Agent



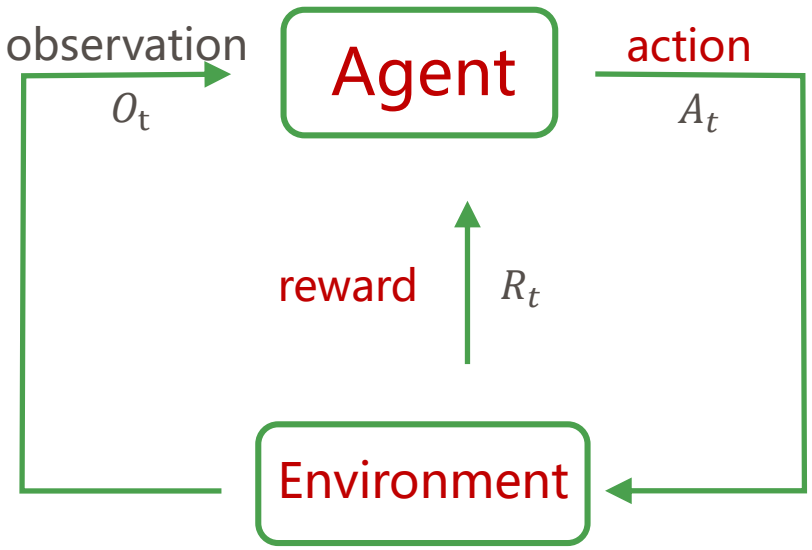
- Major Components of an RL Agent :

  1.Policy: agent’ s behavior function

  2.Value function: how good is each state and/or action

  3.Model: agent’ s representation of the environment

Major Components of a RL Agent

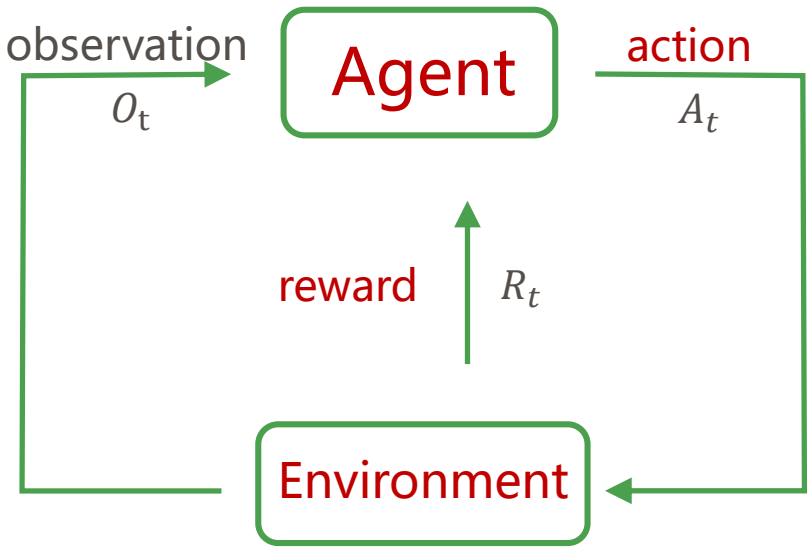


- Policy:
  1. A policy is the agent' s behavior.
  2. It is a map from state to action, e.g.
  3. Deterministic policy: a = $\pi(s)$
  4. Stochastic policy: $\pi(a \mid s) = P[A_t = a | S_t = s]$
- Value Function:
  1. Value function is a prediction of future reward.

$$v_\pi(s) = E_\pi[r_{future} | S_t = s]$$

  2. Used to evaluate the goodness/badness of states.
  3. And therefore to select between actions.

Major Components of a RL Agent



- Model:
  1.A model predicts what the environment will do next.

  2.P predicts the next state.

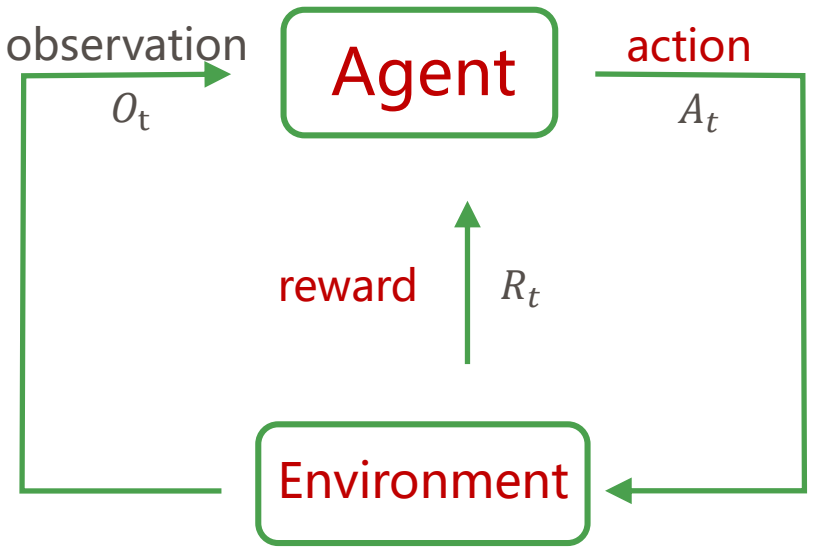
  $$P^a_{ss'} = P[S_{t+1} = s' | S_t = s, A_t = a]$$

  3.R predicts the next (immediate) reward

  $$R^a_s = E[R_{t+1} | S_t = s, A_t = a]$$

- 在模型已知的环境中学习称为“有模型学习”（model-based learning），反之称为“免模型学习”（model-free learning).

RL’ s Object Function



$$\pi^* = arg\max_{\pi} V^{\pi}(s), (\forall s)$$

Review Reinforcement Learning

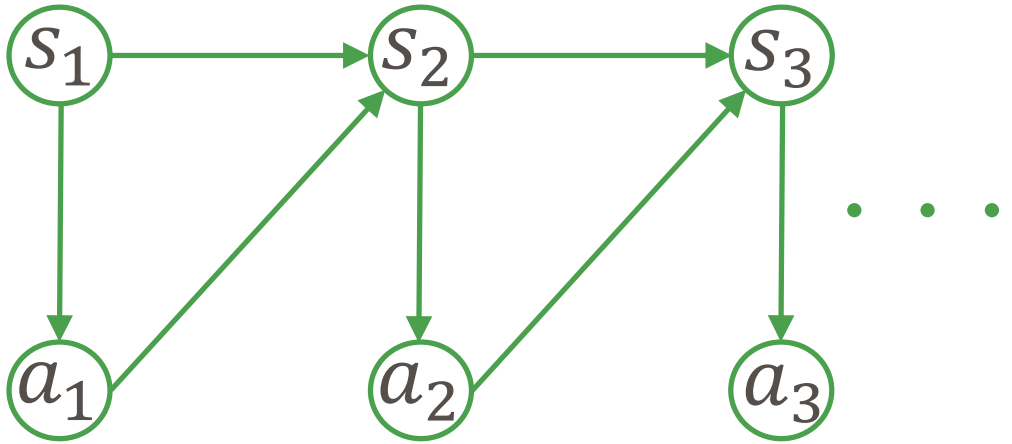
RL in AlphaGo

评价体系　　仿真搜索　　经验转化

Different versions of AlphaGo

AlphaGo Lee　　AlphaGo Zero

RL in AlphaGo



简化：

- 状态集S完全可观察且有限
- 具有马尔可夫性
- 动作集A已知且有限
- 状态转移矩阵（$P_{ss'}^{a}$）已知
- 回报R_s^a已知

评价体系　　仿真搜索　　经验转化

RL in AlphaGo

状 态　　动 作　　策 略

评 价 体 系

RL in AlphaGo 评价体系

Value function is a prediction of future reward. $\boldsymbol{v_\pi(s) = E_\pi[r_{future}|S_t = s]}$

- 采用策略π的情况下未来有限h步的期望立即回报总和

$$V^{\pi}(\mathrm{s})=E_{\pi}\left[\sum_{i=0}^{h} r_i \middle| s_0 = s\right]$$

- 采用策略π的情况下期望的平均回报:

$$V^{\pi}(\mathrm{s})=\lim_{h\to\infty} E_{\pi}\left[\frac{1}{h}\sum_{i=0}^{h} r_i \middle| s_0 = s\right]$$

- 值函数最常见的形式:

$$V^{\pi}(\mathrm{s})=E_{\pi}\left[\sum_{i=0}^{\infty} \gamma^i r_i \middle| s_0 = s\right]$$

式中γ∈[0,1]称为折合因子，表明了未来的回报相对于当前回报的重要程度。
特别的，γ=0时，相当于只考虑立即不考虑长期回报，γ=1时，将长期回报和立即回报看得同等重要。

RL in AlphaGo 评价体系

将值函数的第三种形式展开，其中 r_i 表示未来第 I 步回报，s'表示下一步状态，则有：

$$V^{\pi}(s) = E_{\pi}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \ldots \middle| s_0 = s\right]$$
$$= E_{\pi}\left[r_0 + \gamma E\left[\gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \ldots\right] \middle| s_0 = s\right]$$
$$= E_{\pi}\left[r(s' \mid s, a) + \gamma V^{\pi}(s') \middle| s_0 = s\right]$$

给定策略π和初始状态s，则动作a=π(s)，下个时刻将以概率p(s'|s,a)转向下个状态s'，那么上式的期望可以拆开，可以重写为：

$$V^{\pi}(s) = \sum_{s' \in S} p(s' \mid s, a)\left[r(s' \mid s, a) + \gamma V^{\pi}(s')\right]$$

上面提到的值函数称为状态值函数(state value function)，需要注意的是，在$V^{\pi}(s)$中，π和初始状态s是我们给定的，而初始动作a是由策略π和状态s决定的，即a=π(s)。

RL in AlphaGo

状 态　　动 作　　策 略

评 价 体 系

RL in AlphaGo 评价体系

动作值函数（action value function Q函数）如下：

$$Q^{\pi}(s,a)=E\left[\sum_{i=0}^{\infty}\gamma^{i}r_{i}\middle|s_{0}=s,a_{0}=a\right]$$

给定当前状态s和当前动作a，在未来遵循策略π，那么系统将以概率p(s’ |s,a)转向下个状态s’ ，上式可以重写为：

$$Q^{\pi}(s,a)=\sum_{s'\in S}p(s'\mid s,a)\left[r(s'\mid s,a)+\gamma V^{\pi}(s')\right]$$

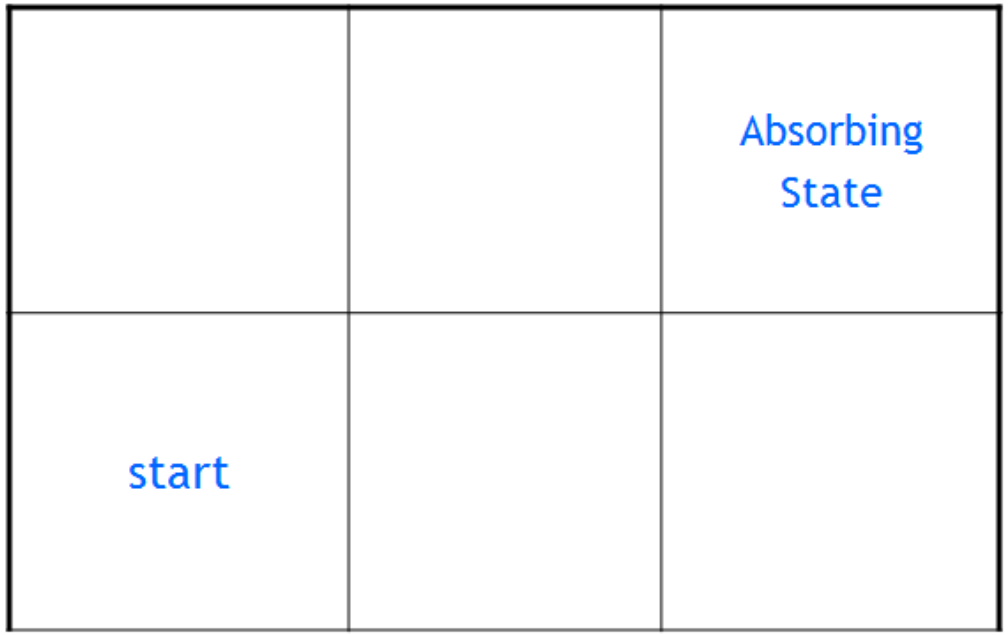
在$Q^{\pi}(s,a)$中，不仅策略π和初始状态s是我们给定的，当前的动作a也是我们给定的，这是$Q^{\pi}(s,a)$和$V^{\pi}(a)$的主要区别。

RL in AlphaGo

状 态　　动 作　　策 略

评 价 体 系

选择方法[值函数与Q函数计算示例]



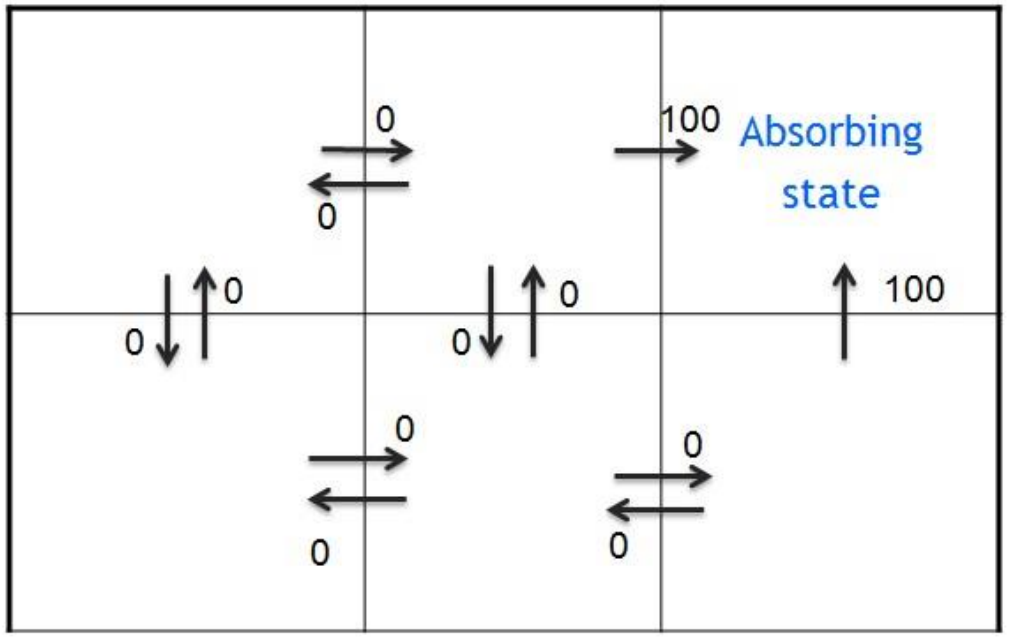
假设agent从左下角的start点出发，右上角为目标位置，称为吸收状态(Absorbing state)，对于进入吸收态的动作，我们给予立即回报100，对其他动作则给予0回报，折合因子γ的值我们选择0.9。

为了方便描述，记第i行，第j列的状态为s_{ij}，在每个状态，有四种上下左右四种可选的动作，分别记为a_u, a_d, a_l, a_r.（up，down，left，right首字母），并认为状态按动作a选择的方向转移的概率为1。

由于状态转移概率是1，每组(s, a)对应了唯一的s'。回报函数r(s ‘ | s , a)可以简记为r(s, a).

值函数与Q函数计算示例

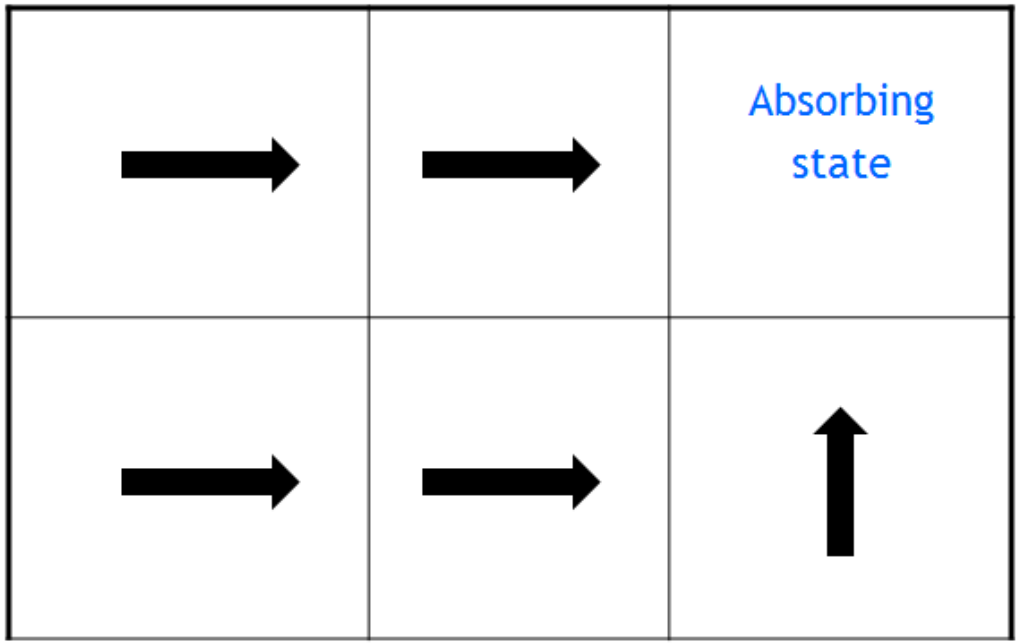
选择方法[值函数与Q函数计算示例]



如下所示，每个格子代表一个状态s，箭头则代表动作a，旁边的数字代表立即回报，可以看到只有进入目标位置的动作获得了回报100，其他动作都获得了0回报。 即$r(s_{12},a_r) = r(s_{23},a_u) = 100$。

值函数与Q函数计算示例

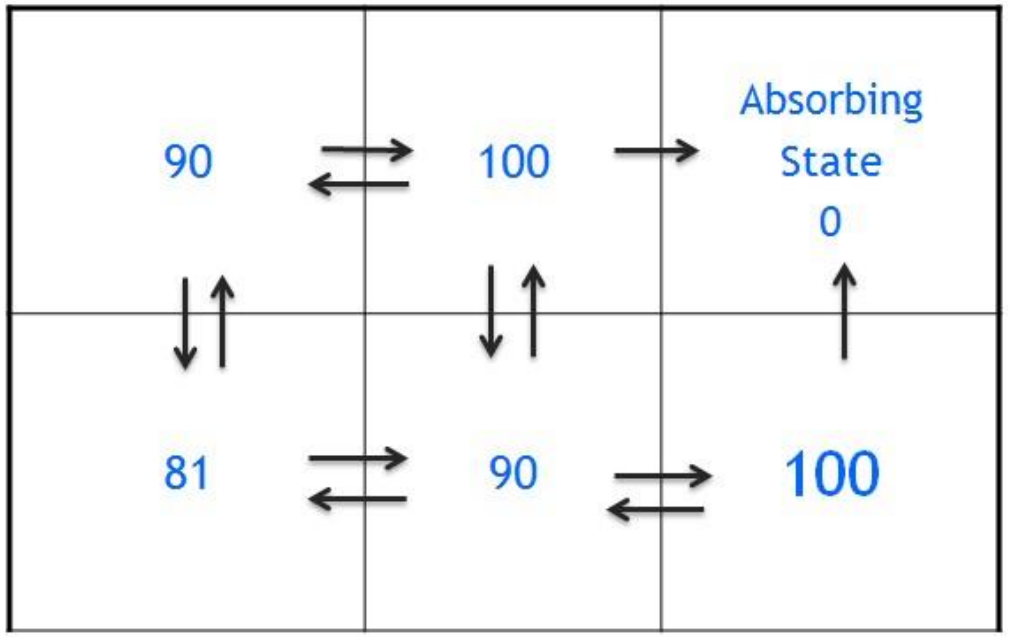
选择方法[值函数与Q函数计算示例]



一个策略π如左图所示

值函数与Q函数计算示例

选择方法[值函数与Q函数计算示例]



值函数$V^{\pi}(s)$如下所示:

根据V^{π}的表达式，立即回报，和策略π，有

$V^{\pi}(s_{12}) = r(s_{12},a_r) = r(s_{13}|s_{12},a_r) = 100$

$V^{\pi}(s_{11}) = r(s_{11},a_r)+\gamma*V^{\pi}(s_{12}) = 0+0.9*100 = 90$

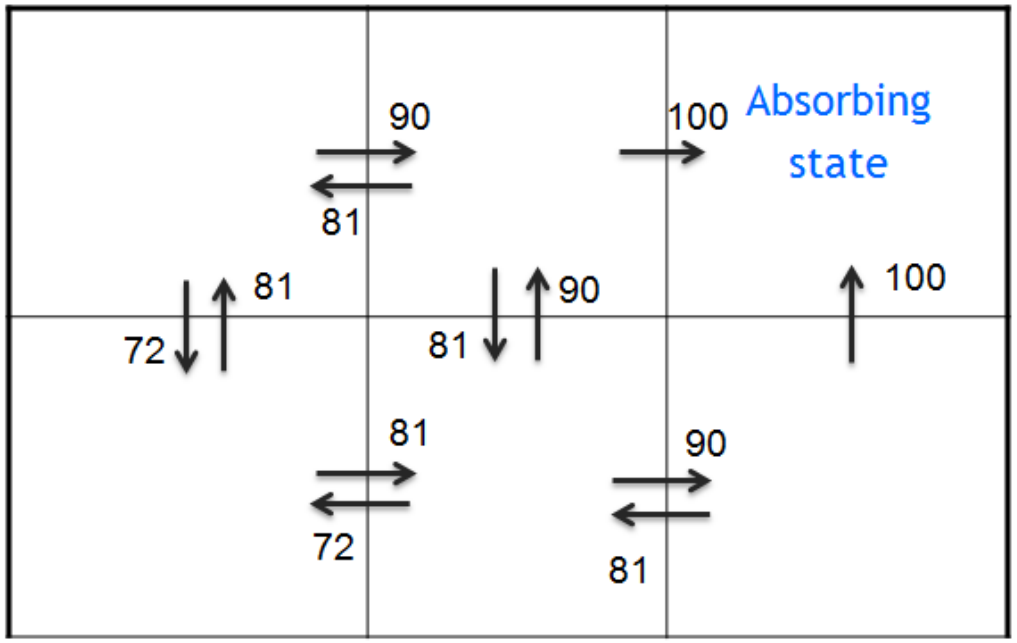
$V^{\pi}(s_{23}) = r(s_{23},a_u) = 100$

$V^{\pi}(s_{22}) = r(s_{22},a_r)+\gamma*V^{\pi}(s_{23}) = 90$

$V^{\pi}(s_{21}) = r(s_{21},a_r)+\gamma*V^{\pi}(s_{22}) = 81$

$$V^{\pi}(s)=\sum_{s'\in S} p(s'|s,a)\left[r(s'|s,a)+\gamma V^{\pi}(s')\right]$$

选择方法[值函数与Q函数计算示例]



Q(s,a)值如下所示:

对s_{11}计算Q函数（用到了上面V^{π}的结果）如下:

$Q^{\pi}(s_{11},a_r)=r(s_{11},a_r)+\gamma * V^{\pi}(s_{12})$

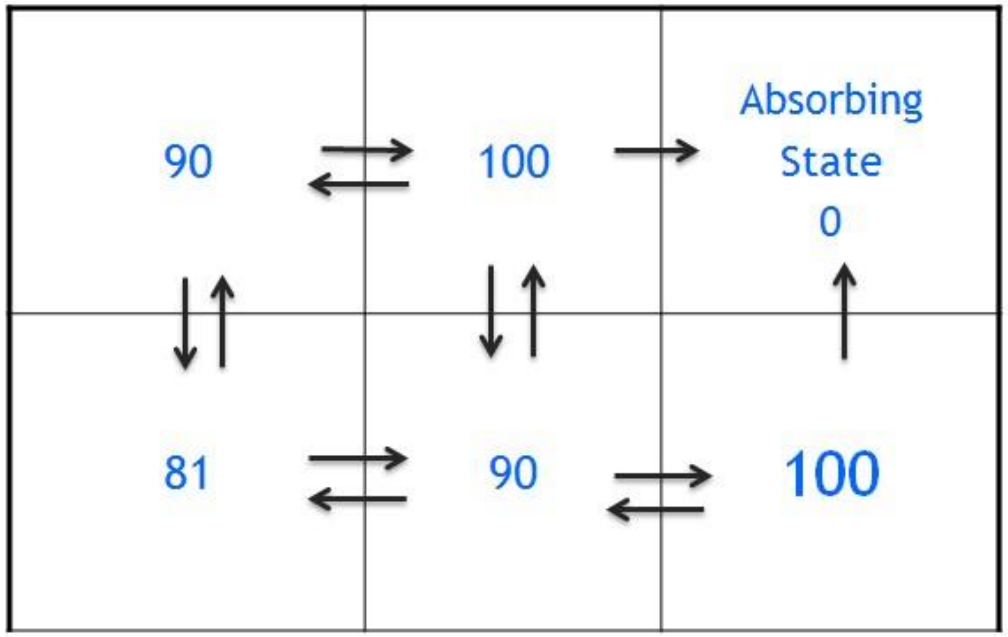
$=0+0.9*100 = 90$

$Q^{\pi}(s_{11},a_d)=r(s_{11},a_d)+\gamma * V^{\pi}(s_{21}) = 72$

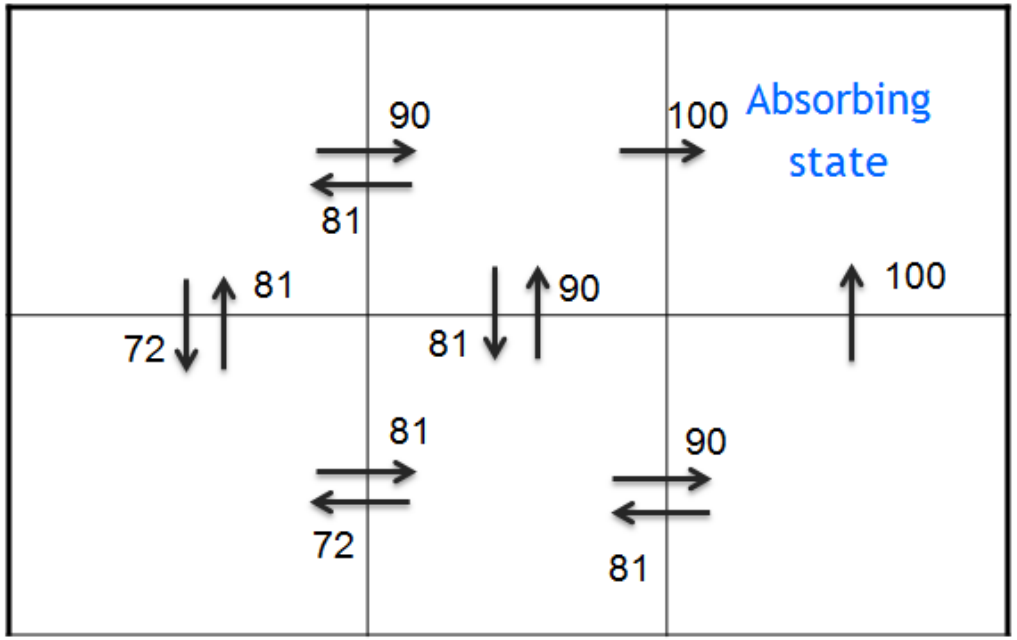
$$Q^{\pi}(s,a)=\sum_{s'\in S} p(s'|s,a)\left[r(s'|s,a)+\gamma V^{\pi}(s')\right]$$

选择方法[值函数与Q函数计算示例]

V函数



Q函数



思考：此例中的最优策略什么？怎么得到？

RL in AlphaGo 评价体系

策略评价:

Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

策略选择的动作改变能否为当前最优的动作?

RL in AlphaGo 评价体系

优化目标π*可以表示为： $\pi^*(s) = \arg\max_{\pi} V^{\pi}(s)$

分别记最优策略π*对应的状态值函数和行为值函数为V*(s)和Q*(s, a)，由它们的定义容易知道，V*(s)和Q*(s, a)存在如下关系:

$$V^*(s) = \max_a Q^*(s, a)$$

即状态值函数和行为值函数分别满足如下贝尔曼最优性方程(Bellman optimality equation)：

$$V^*(s) = \max_a E\left[r(s'|s,a) + \gamma V^*(s') \middle| s_0 = s\right]$$

$$= \max_{a \in A(s)} \sum p(s'|s,\pi(s))\left[r(s'|s,\pi(s)) + \gamma V^{\pi}(s')\right]$$

$$Q^*(s) = E\left[r(s'|s,a) + \gamma \max_{a'} Q^*(s',a') \middle| s_0 = s, a_0 = a\right]$$

$$= \sum p(s'|s,\pi(s))\left[r(s'|s,\pi(s)) + \gamma \max_{a \in A(s)} Q^*(s',a')\right]$$

RL in AlphaGo 评价体系

贝尔曼方程(Bellman equation)将“决策问题在特定时间点的值以来自初始选择的报酬及由初始选择衍生的决策问题的值”的组合形式表示。以此将原问题变成较简单的子问题，子问题遵守由贝尔曼提出的最优化原理。

最优化原理：一个最优策略，具有如下性质：不论初始状态和初始决策如何，以第一步决策所形成的阶段和状态作为初始条件来考虑时，余下的决策对余下的问题而言也比构成最优策略。

RL in AlphaGo 评价体系

贝尔曼最优性方程揭示了非最优策略的改进方式：将策略选择的动作改变为当前最优的动作。

不妨令动作改变后对应的策略记为π'，改变动作的条件为$Q^{\pi}(x,\pi'(x)) \geq V^{\pi}(x)$，以$\gamma$折扣累积奖赏为例

$$\begin{aligned} V^{\pi}(x) &\leqslant Q^{\pi}(x,\pi'(x)) \\ &= \sum_{x'\in X} P_{x\to x'}^{\pi'(x)}(R_{x\to x'}^{\pi'(x)} + \gamma V^{\pi}(x')) \\ &\leqslant \sum_{x'\in X} P_{x\to x'}^{\pi'(x)}(R_{x\to x'}^{\pi'(x)} + \gamma Q^{\pi}(x',\pi'(x'))) \end{aligned}$$

值函数对于策略的每一点改进都是单调递增的（由单调有界收敛性定理知最优策略总是存在的），因此对于当前策略π，可将其改进为

$$\pi'(x) = \arg\max_{a\in A} Q^{\pi}(x,a),$$

直到π'和π一致，就找到了最优策略。

RL in AlphaGo 评价体系

策略估计(Policy Evaluation)

首先，对于任意的策略π，我们如何计算其状态值函数$V^{\pi}(s)$？这个问题被称作策略估计，
前面讲到对于确定性策略，值函数

现在扩展到更一般的情况，如果在某策略π下，π(s)对应的动作a有多种可能，每种可能记为π(a|s)，则状态值函数定义如下：

一般采用迭代的方法更新状态值函数，首先将所有$V^{\pi}(s)$的初值赋为0（其他状态也可以赋为任意值，不过吸收态必须赋0值），然后采用如下式子更新所有状态s的值函数（第k+1次迭代）：

用一个数组保存各状态值函数，每当得到一个新值，就将旧的值覆盖,形如$[V^{k+1}(s_1),V^{k+1}(s_2),V^{k}(s_3)...]$ 。

RL in AlphaGo 评价体系

策略估计(Policy Evaluation)

整个策略估计算法如下图所示

Input π, the policy to be evaluated
Initialize an array $v(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $temp \leftarrow v(s)$
    $v(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v(s')]$
    $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $v \approx v_\pi$

RL in AlphaGo 评价体系

策略改进(Policy Improvement)

- 策略估计的目的，是为了寻找更好的策略，用关于值函数的贪心策略获得新策略，改进旧策略的过程叫做策略改进(Policy Improvement)
- 评判标准是：$Q^{\pi}(s,a)$是否大于$V^{\pi}(s)$。如果$Q^{\pi}(s,a) > V^{\pi}(s)$，那么至少说明新策略【仅在状态s下采用动作a，其他状态下遵循策略π】比旧策略【所有状态下都遵循策略π】整体上要更好。
- 采用贪心策略来获得新策略π'：

$$\pi'(s) = \arg\max_{a} Q^{\pi}(s,a)$$

$$= \arg\max_{a} E_{\pi}\left[r(s' \mid s,a) + \gamma V^{\pi}(s') \middle| s_0 = s, a_0 = a \right]$$

$$= \arg\max_{a} \sum_{s' \in S} p(s' \mid s,a)\left[r(s' \mid s,a) + \gamma V^{\pi}(s') \right]$$

RL in AlphaGo 评价体系

策略迭代(Policy Iteration)

假设我们有一个策略π，那么我们可以用policy evaluation获得它的值函数$V^{\pi}(s)$，然后根据policy improvement得到更好的策略π'，接着再计算$V^{\pi'}(s)$,再获得更好的策略π''，整个过程顺序进行如下图所示：

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

RL in AlphaGo 评价体系

策略迭代（Policy Iteration)

右图给出的算法描述是策略评估的基础上加入策略改进而形成的策略迭代算法。

1. Initialization
   $v(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
      $\Delta \leftarrow 0$
      For each $s \in \mathcal{S}$:
         $temp \leftarrow v(s)$
         $v(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) \left[r(s, \pi(s), s') + \gamma v(s') \right]$
         $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

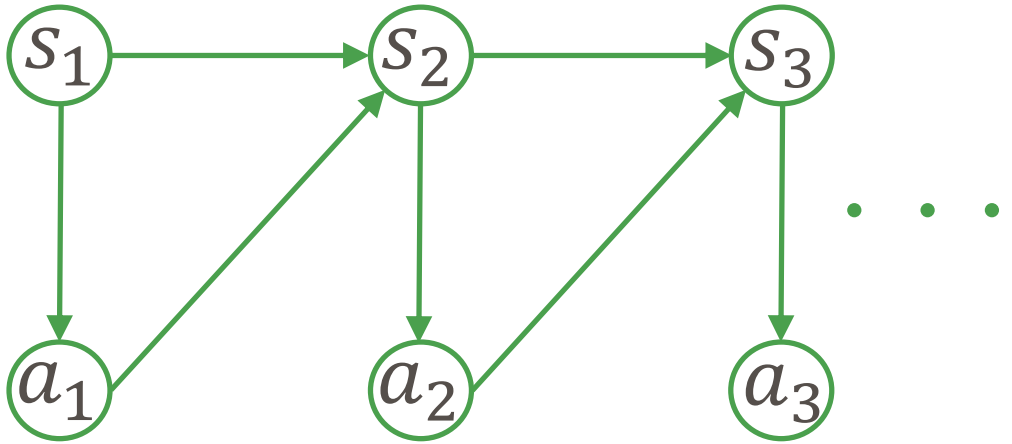
3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
      $temp \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s, a) \left[r(s, a, s') + \gamma v(s') \right]$
      If $temp \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If *policy-stable*, then stop and return v and π; else go to 2

RL in AlphaGo 评价体系

状态转移概率$P_{ss'}^{a}$　　即时惩罚R_{s}^{a}　　状态值函数$V^{\pi}(s)$

评估动作并选择当前最大回报

RL in AlphaGo 仿真搜索



特点

- 状态集S过大
- 动作集A过大
- 状态转移概率$P_{ss'}^{a}$,无模型，难获得
- 回报R_s^a延迟过长，难估计

评价体系　　**仿真搜索**　　经验转化

RL in AlphaGo

蒙特卡洛方法　　树搜索　　自对弈

仿 真 搜 索

AlphaGo

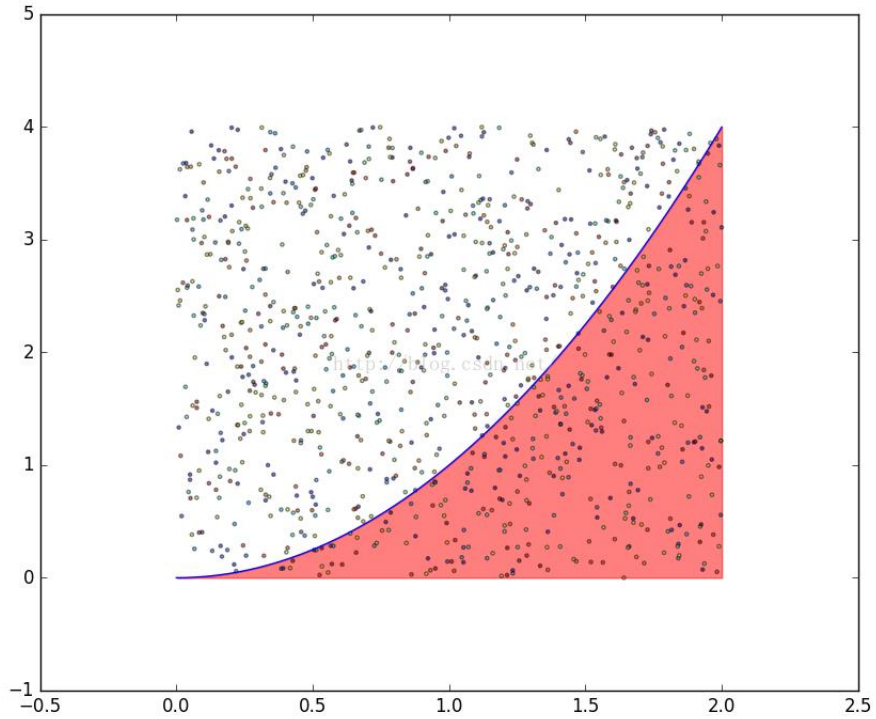
RL in AlphaGo 仿真搜索

在现实的强化学习任务中，环境的转移概率、奖赏函数往往很难得知，甚至很难知道环境中一共有多少状态。如果学习算法不依赖于环境建模，则成为“免模型学习”(model-free learning)。在免模型情形下，策略迭代算法由于模型未知使得策略评估无法全概率展开而难以估计。

用多次“采样”，求平均累积奖赏来作为期望累积奖赏的近似，称为蒙特卡洛强化学习。

在初始状态s，遵循策略π，最终获得了总回报R，这就是一个样本。如果我们有许多这样的样本，就可以估计在状态s下，遵循策略π的期望回报，也就是状态值函数$V^{\pi}(s)$了。蒙特卡罗方法就是依靠样本的平均回报来解决增强学习问题的。

RL in AlphaGo 仿真搜索



求函数$y=x^2$在[0,2]区间的积分

蒙特卡洛方法（Monte Carlo method）是以概率统计理论、方法为基础的一种数值计算方法，将所求解的问题同一定的概率模型相联系，用计算机实现统计模拟或抽样，以获得问题的近似解，故又称随机抽样法或统计试验法 。

蒙特卡洛方法可以应用在很多场合，但求的是近似解，在模拟样本数越大的情况下，越接近与真实值，但样本数增加会带来计算量的大幅上升。

蒙特卡洛方法的理论基础是大数定理。

蒙特卡洛方法本身不是优化方法 。那它如何来求解优化问题呢?

RL in AlphaGo 仿真搜索

蒙特卡洛方法（Monte Carlo Method)

蒙特卡罗方法可以求面积；

而（二元）积分是计算曲线围成的（带符号）面积，
所以蒙特卡罗方法可以求积分；

而期望就是概率密度与随机变量乘积的积分，
所以蒙特卡罗方法可以求期望；

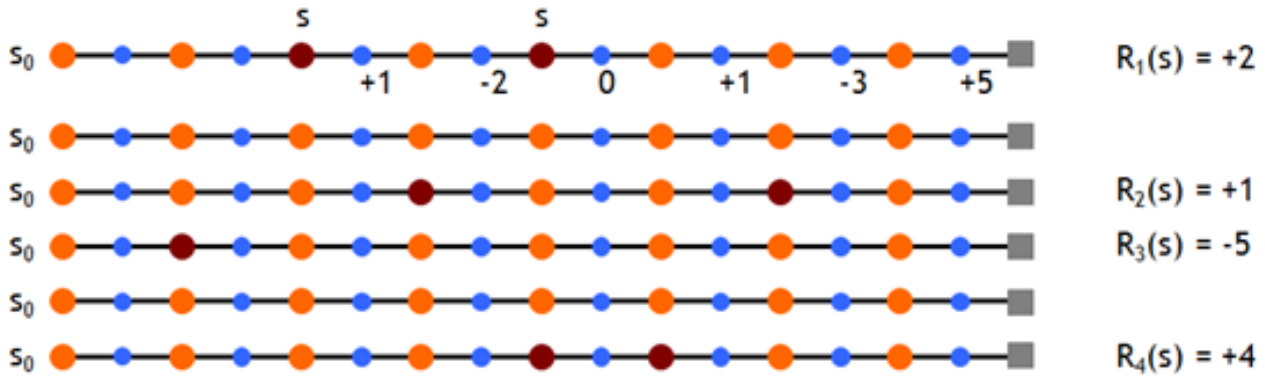
而最优化问题的解往往是期望或者与期望相关的函数，
所以蒙特卡罗方法常常可以求最优解……

RL in AlphaGo 仿真搜索

蒙特卡洛策略估计（Monte Carlo Policy evaluation)

首先考虑用蒙特卡罗方法来学习状态值函数$V^{\pi}(s)$。我们仅将蒙特卡罗方法定义在episode task上，所谓的episode task就是指不管采取哪种策略π，都会在有限时间内到达终止状态并获得回报的任务。比如玩棋类游戏，在有限步数以后总能达到输赢或者平局的结果并获得相应回报

现在我们假设有如下一些样本，取折扣因子γ=1，即直接计算累积回报，则有
$V^{\pi}(s) \approx (2 + 1 - 5 + 4)/4 = 0.5$ (只记录s的第一次访问)



根据大数定理容易知道，当我们经过无穷多的episode后，$V^{\pi}(s)$的估计值将收敛于其真实值。

RL in AlphaGo 仿真搜索

动作值函数的MC估计（MC Estimation of Action Values)

$Q^{\pi}(s,a)$的估计方法与前面类似，即在状态s下采用动作a，后续遵循策略π获得的期望累积回报用平均回报来估计，即为$Q^{\pi}(s,a)$。即模型未知情况下，我们使用某种策略进行采样，获得采样轨迹（history）

$$H_t = O_1, R_1, A_1, \ldots\ldots, A_{t-1}, O_t, R_t$$

然后对轨迹中出现的每一对状态-动作，记录其后的奖赏之和，作为状态-动作对的一次累积奖赏采样值。多次采样之后求取平均奖赏，得到动作值函数的估计。

然而这里存在一个问题：我们的策略有可能是确定性的，即对于某个状态只会输出一个动作，若使用这样的策略进行采样，则只能得到多条相同的轨迹（history），使得动作值函数和状态值函数都无法有效更新。

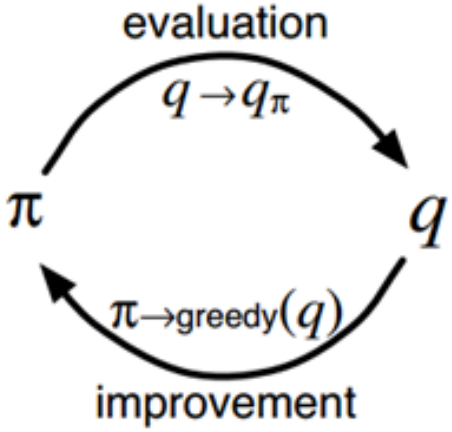
RL in AlphaGo 仿真搜索

持续探索（Maintaining Exploration)

Maintaining Exploration的思想很简单，即在所有的状态下，用1-ε的概率来执行当前的最优动作，ε的概率来执行其他动作。这样我们就可以获得所有动作的估计值，然后通过慢慢减少ε值，最终使算法收敛，并得到最优策略。

MC版本的策略迭代过程

$$\pi_0 \xrightarrow{\text{E}} q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_*$$



RL in AlphaGo 仿真搜索

策略迭代（Policy Iteration）

输入： 环境 E;
动作空间 A;
起始状态 x_0;
策略执行步数 T.

过程：

1: $Q(x,a)=0,\ \mathrm{count}(x,a)=0,\ \pi(x,a)=\frac{1}{|A(x)|}$;
2: **for** $s=1,2,\ldots$ **do**
3: 在 E 中执行策略 π 产生轨迹
$< x_0,a_0,r_1,x_1,a_1,r_2,\ldots,x_{T-1},a_{T-1},r_T,x_T >$;
4: **for** $t=0,1,\ldots,T-1$ **do**
5: $R=\frac{1}{T-t}\sum_{i=t+1}^{T} r_i$;
6: $Q(x_t,a_t)=\frac{Q(x_t,a_t)\times\mathrm{count}(x_t,a_t)+R}{\mathrm{count}(x_t,a_t)+1}$;
7: $\mathrm{count}(x_t,a_t)=\mathrm{count}(x_t,a_t)+1$
8: **end for**
9: 对所有已见状态 x :

$$\pi(x,a)=\begin{cases}\arg\max_{a'} Q(x,a'), & \text{以概率 } 1-\epsilon;\\ \text{以均匀概率从 } A \text{ 中选取动作}, & \text{以概率 } \epsilon .\end{cases}$$

10: **end for**

输出： 策略 π

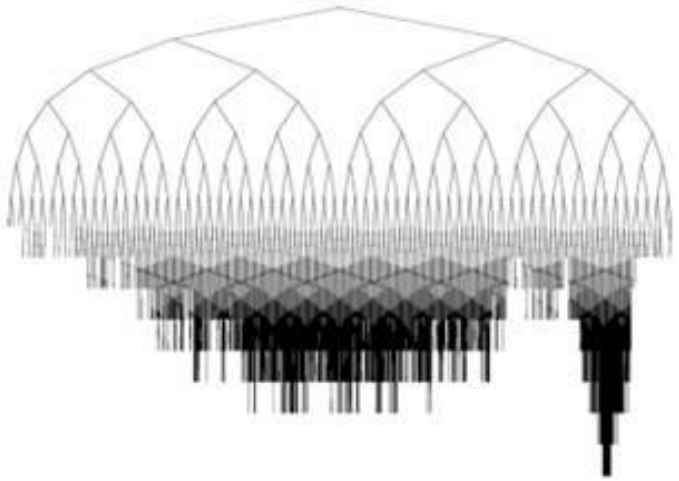
RL in AlphaGo

蒙特卡洛方法　　树搜索　　自对弈

仿 真 搜 索

RL in AlphaGo 仿真搜索

MCTS



蒙特卡洛搜索树（Monte-Carlo Tree Search）

用均匀分布初始化策略，模拟对弈双方的走法，当分出胜负的时候，给胜方轨迹的所有状态动作对加一分（初始为零分）并根据新分数实时更新动作状态对的概率，大量模拟对弈之后，获胜概率高的落子方法就会“涌现”出来。

特点：

没有人工设计的特征，完全依靠规则本身，通过不断自对弈来提高能力。

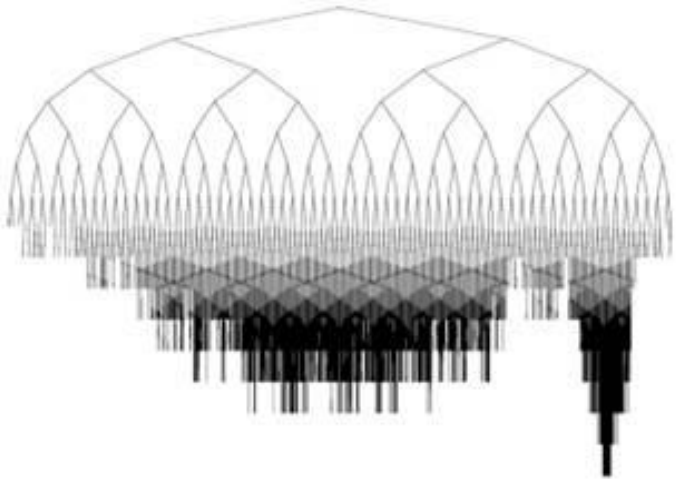
在对手思考对策的同时自己也可以思考对策，比较仿生。

缺点：

初始策略太简单，学习效率低。

RL in AlphaGo 仿真搜索

MCTS



可以通过两个原则将有效搜索空间缩小：

第一个原则是通过位置评估**降低树的深度**：在状态s的时候将树截断，通过估计一个可以对从状态s开始的游戏胜负结局进行预测的价值函数：$v(s) = v^*(s)$来将状态s以下的子树代替。

第二个原则是通过从一个策略$p(a|s)$进行动作采样来**减少搜索树的宽度**，其中$p(a|s)$是一个在状态s时的可行动作的概率分布。

RL in AlphaGo

蒙特卡洛方法　　树搜索　　自对弈

仿 真 搜 索

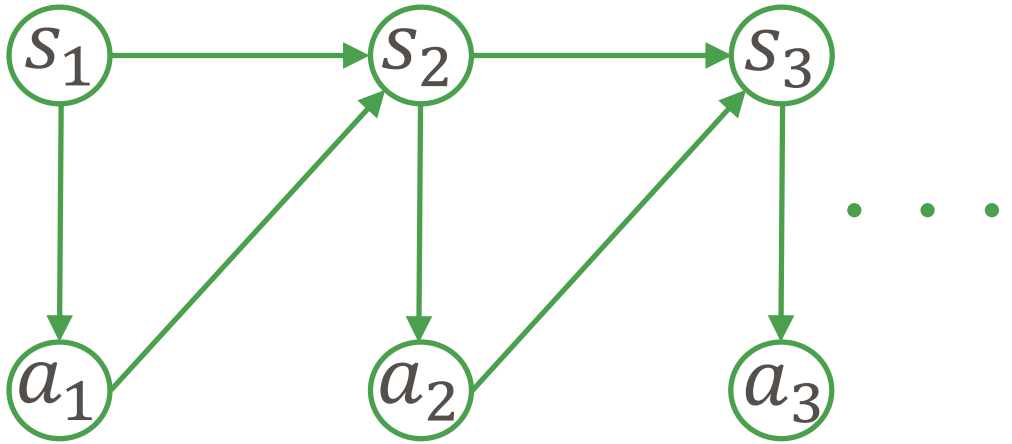
RL in AlphaGo 自对弈

产生新样本

性能迭代提升

可以嵌入任何解决方案

RL in AlphaGo



特点

- 状态集S过大
- 动作集A过大
- 状态转移概率$P_{ss'}^{a}$,无模型，难获得
- 回报R_s^a延迟过长，难估计

评价体系　　仿真搜索　　**经验转化**

RL in AlphaGo

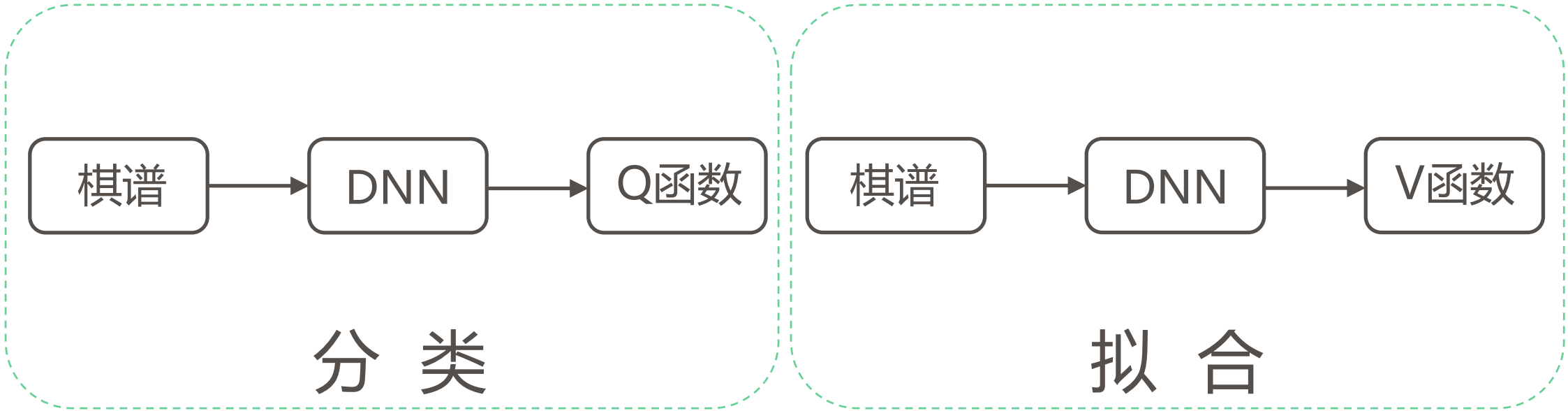
动作值函数$Q^{\pi}(s)$

状态值函数$V^{\pi}(s)$

经 验 转 化

AlphaGo

RL in AlphaGo 经验转化



嵌入MCTS

Review Reinforcement Learning

RL in AlphaGo

评价体系　　仿真搜索　　经验转化

Different versions of AlphaGo

AlphaGo Lee　　AlphaGo Zero

AlphaGo Fan -AlphGo Zero 版本差异

- AlphaGo Fan在2015年10月以5:0打败了欧洲围棋冠军樊麾。
- AlphaGo Lee在2016年3月以4:1战胜了职业九段棋手李世石。
- AlphaGo Master 在2016年12月29日至2017年1月4日，再度强化的AlphaGo以“Master”为账号名称，借助非正式的网络快棋对战进行测试，挑战中韩日台的一流高手，60全胜；2017年5月，AlphaGo Master对战世界第一棋手柯洁，获得3:0全胜的战绩。
- AlphaGo Zero 2017年10月，Google旗下的子公司Deep mind的发布了名为《Mastering the game of Go without human knowledge》的论文，介绍了AlphaGo最强版本AlphaGo Zero。

AlphaGo Fan -AlphGo Zero 版本差异

- Alpha Go Fan和Alpha Go Lee在算法、神经网络架构等方面相似。
- AlphaGo Fan通过策略网络对弈的数据训练价值网络；AlphaGo Lee的价值网络是以Alpha Go的快速自我对弈的数据进行训练的，
- Alpha Go Lee中的策略网络和价值网络比Alpha Go Fan中的更大，训练的迭代次数也更多。
- Alpha Go Lee和Alpha Go Fan采用了两类神经网络：策略网络和价值网络，策略网络以当前位置的棋盘状态作为输入，输出在该状态采取每一个动作的概率分布，而价值网络则是对当前的位置进行评估，预测处于当前的位置比赛的胜负结果。Alpha Go Lee和Alpha Go Fan都会采用合适的数据通过监督学习的方式预先训练好快速走子策略网络p_π，然后训练好策略网络p_σ，以便对强化学习的策略网络p_ρ的训练进行初始化，最后借助于p_σ和p_ρ对价值网络$v_\theta(s)$进行回归.在进行比赛的时候,在处于每一个棋盘状态时,借助于之前的各种策略网络以及价值网络的指导,进行蒙特卡洛树搜索,以便决定下一步棋怎么下。.

AlphaGo Fan -AlphGo Zero 版本差异

- AlphaGo Master和AlphaGo Zero使用了相同的神经网络架构、强化学习算法、以及蒙特卡洛树搜索算法，但是AlphaGo Master使用了人工处理的特征以及和AlphaGo Lee类似的快速走子策略，也采用人类棋手对弈的数据通过监督学习的方式进行初始化训练。

- Alpha Go Zero和Alpha Go Master除了训练方式不一样，采用的算法、神经网络架构以及搜索策略都一样，所以主要对Alpha Go Zero进行描述。Alpha Go Zero的训练除了一些围棋的领域知识外，没有采用人类的其他知识，完全从白板开始进行训练，训练的数据通过当前的网络和前面训练产生的网络进行对弈产生。Alpha Go Zero只采用了一个神经网络，之后在该神经网络的指导下，进行蒙特卡洛树搜索，而且利用搜索产生的数据对神经网络进行更新。在进行比赛的时候，处于每一个棋盘状态时，会进行一次蒙特卡洛树搜索，搜索结束返回当前状态采取各种行动的概率分布，之后决定下一步棋该怎么下。

AlphaGo Fan -AlphGo Zero 版本差异

性能配置

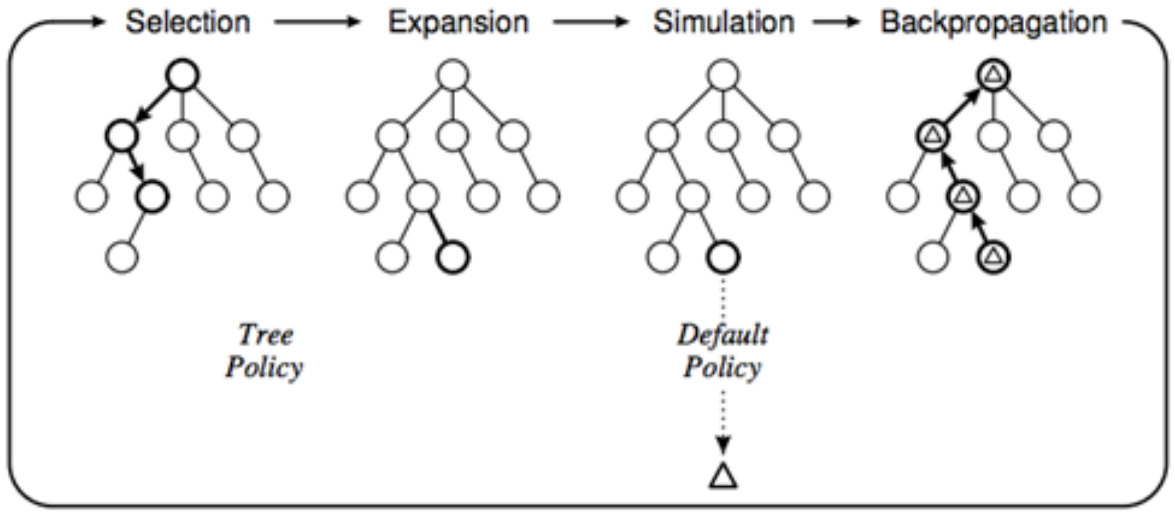
版本	硬件	Elo等级分的理论峰值
Alpha Go Fan	176GPU，分布式	3144
Alpha Go Lee	48个TPU，分布式	3739
Alpha Go Master	4个TPU，单机	4858
Alpha Go Zero	4个TPU，单机	5185

AlphaGo Lee

AlphaGo Lee 演进及原理

DNN+MCTS

以人类棋谱为输入,训练得到策略函数
a = P_human(s)[业余6段],以此做初始化策略，新分数按如下方式更新：
新分数= 初始分+ 通过模拟得到的赢棋概率
然而，P_human()计算太慢了。
我们训练一个网络层数和特征数都减少的简化版策略P_human_fast()。
先用P_human()来开局(迅速缩小搜索空间)，走大概20多步，后面再使用P_human_fast()走到最后（更多探索）。



综合了DNN和MCTS的围棋程序已经可以战胜所有其他电脑，仍然距离人类职业选手仍有不小的差距。怎么办?

AlphaGo Lee 演进及原理

DNN+MCTS

自对弈

先用P_human和P_human对弈，得到比如一万个新棋谱，加入到训练集当中，训练出P_human_1。然后再让P_human_1和P_human_1对局，得到另外一万个新棋谱，这样可以训练出P_human_2，如此往复，可以得到P_human_n。P_human_n得到了最多的训练，棋力比原来更强。我们给最后这个策略起一个新名字：P_human_plus。

这时，再让P_human_plus和P_human对局，在不用任何搜索的情况下胜率可达80%，但代入到MCTS中用P_human_plus来开局，剩下的用P_human_fast。这样棋力反而不如用P_human开局的方式。（初始搜索空间缩得太小，陷入局部最优）

用离线的自对弈方式可以增强P_human，但并不是解决问题真正的出路

RL in AlphaGo 自对弈

产生新样本

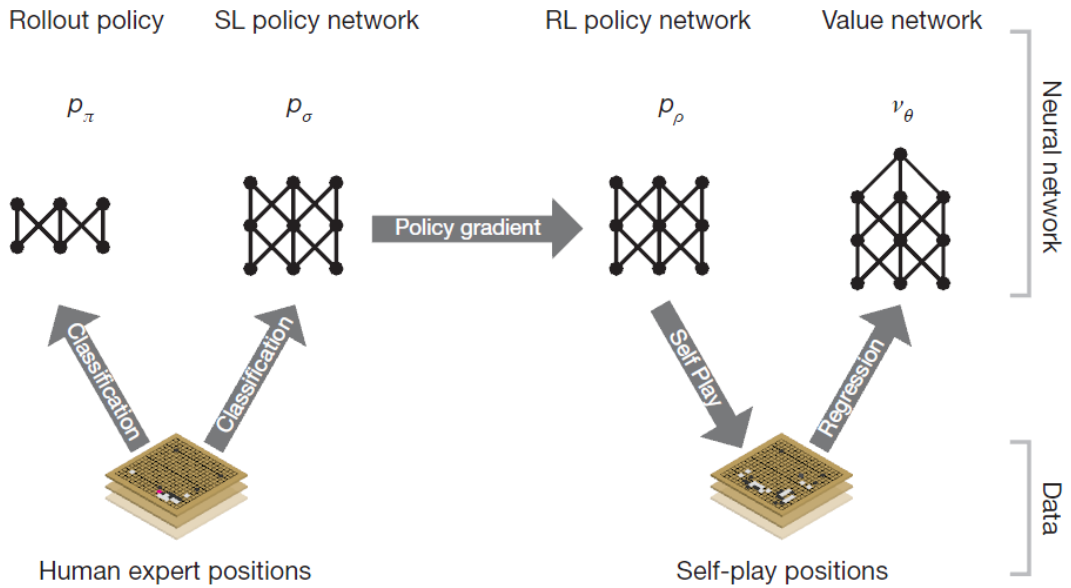
性能迭代提升

可以嵌入任何解决方案

局限（如何解决？）

AlphaGo Lee 演进及原理

DNN+MCTS+状态值函数v()



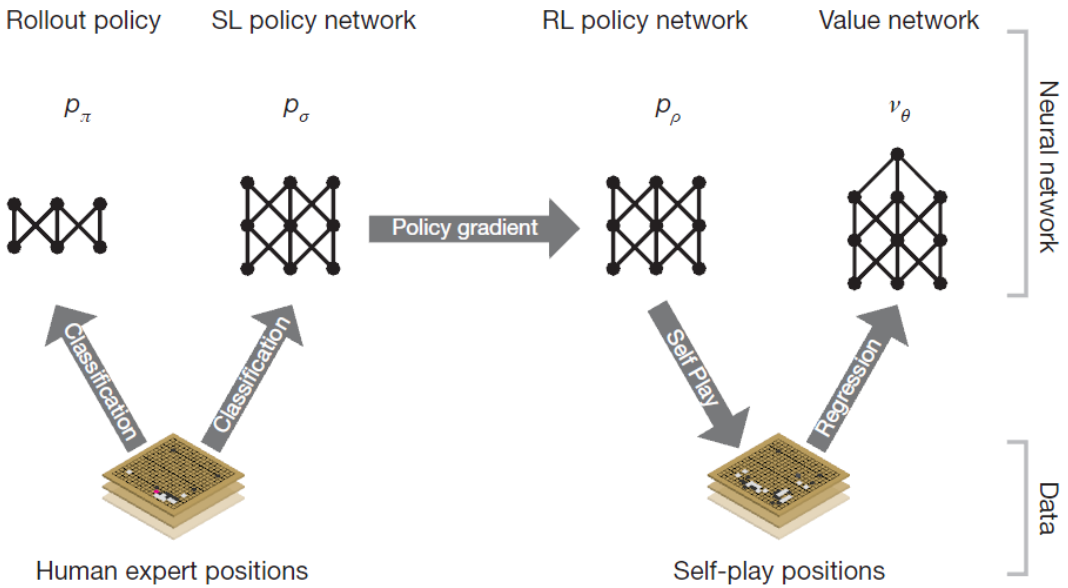
如果能评估局面，就能给MCTS更多的搜索依据。

具体做法：

先用P_human走L步，这样有利于生成更多局面。为了进一步扩大搜索空间，在L+1步的时候，干脆完全随机掷一次骰子，记下这个状态SL+1，然后再用P_human_plus来对弈，直到结束获得结果r。如此不断对弈，由于L也是一个随机数，我们就得到了开局、中盘、官子不同阶段的很多局面s，和这些局面对应的结果r。有了这些训练样本<s,r>，还是使用神经网络，把最后一层的目标改成回归而非分类，就可以得到一个v( )函数，输出赢棋的概率。

AlphaGo Lee 演进及原理

DNN+MCTS+状态值函数v()



- 策略网络包括输入层、11个隐形层和一个输出层
- 价值网络包括输入层、13个隐形层（1至10隐形层和策略网络的一样，11层是一个卷积层，13层是带有256个整流单元的全连接线性层），输出层是一个带有一个tanh单元的全连接线性层。

AlphaGo Lee 演进及原理

DNN+MCTS+状态值函数v()

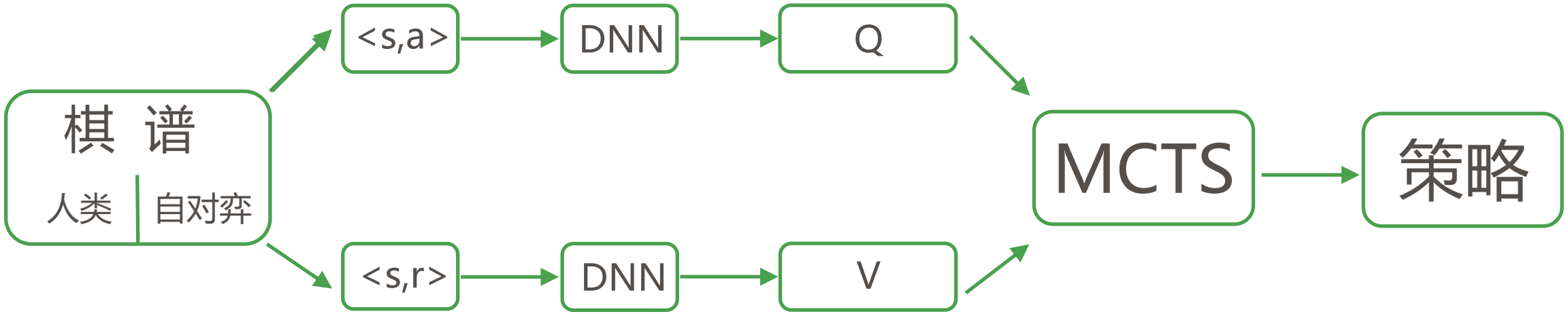
在MCTS框架之上融合局面评估函数v()：

用P_human作为初始分开局，每局选择分数最高的方案落子，下到第L步之后，改用P_human_fast把剩下的棋局走完，同时调用v()，评估局面的获胜概率。然后按照如下规则更新整个树的分数：

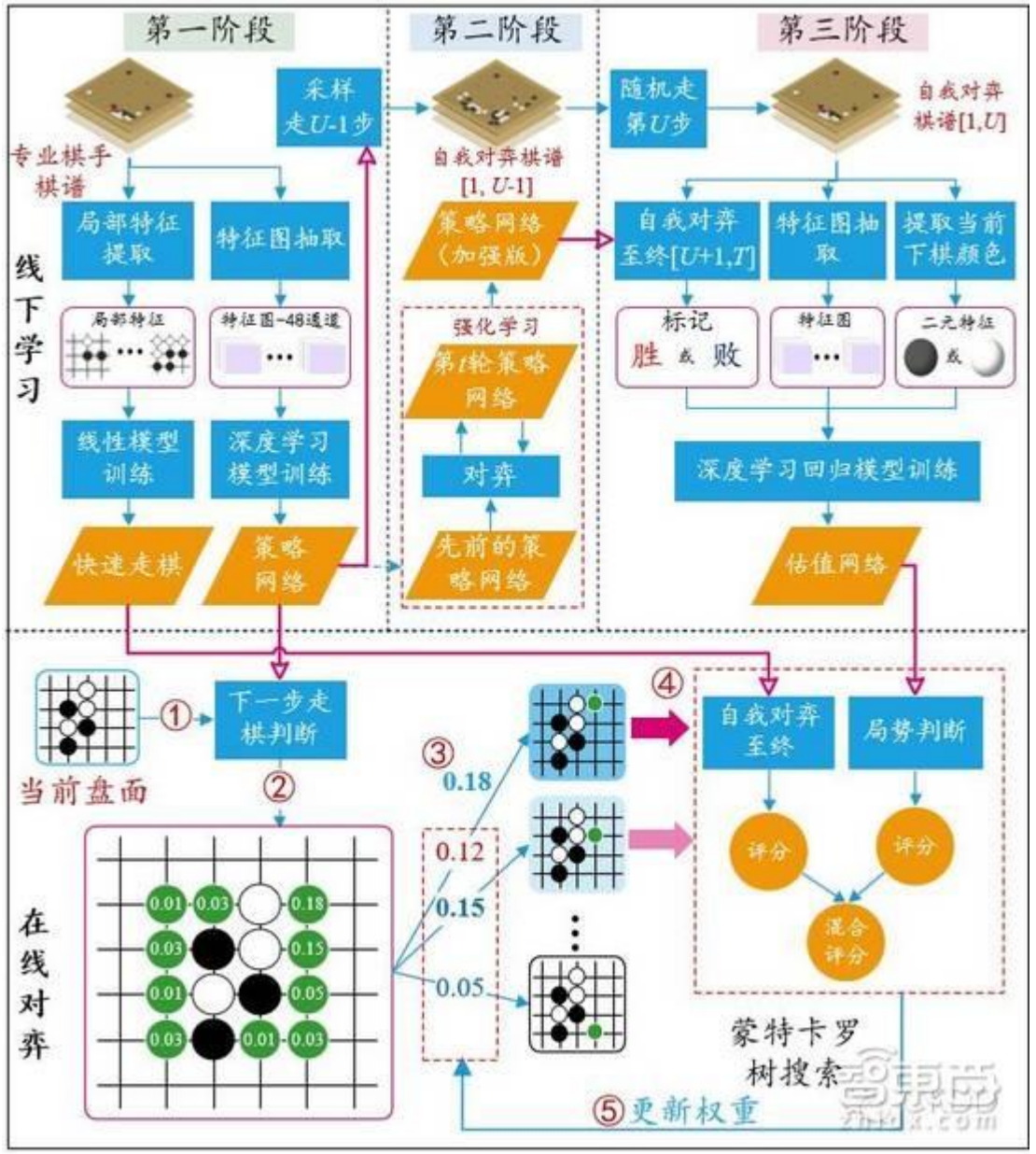
新分数= 调整后的初始分+ 0.5 * 通过模拟得到的赢棋概率 + 0.5 * 局面评估分

如果v()表示大局观，“P_human_fast模拟对局”表示快速验算，那么上面的方法就是大局观和快速模拟验算并重。

AlphaGo Lee 演进及原理



AlphaGo Lee


第一阶段
第二阶段
第三阶段
专业棋手棋谱
采样走U-1步
自我对弈棋谱[1, U-1]
随机走第U步
自我对弈棋谱[1,U]
线下学习
局部特征提取
特征图抽取
策略网络（加强版）
自我对弈至终[U+1,T]
特征图抽取
提取当前下棋颜色
局部特征
特征图-48通道
强化学习
标记
胜 败
特征图
二元特征
线性模型训练
深度学习模型训练
第t轮策略网络
对弈
深度学习回归模型训练
快速走棋
策略网络
先前的策略网络
估值网络
当前盘面
①
下一步走棋判断
②
③
④
0.18
0.12
0.15
0.05
自我对弈至终
局势判断
评分
评分
混合评分
在线对弈
0.01 0.03 0.18
0.03 0.15
0.01 0.05
0.03 0.01 0.03
蒙特卡罗树搜索
⑤更新权重

AlphaGo Zero

AlphaGo Zero 原理

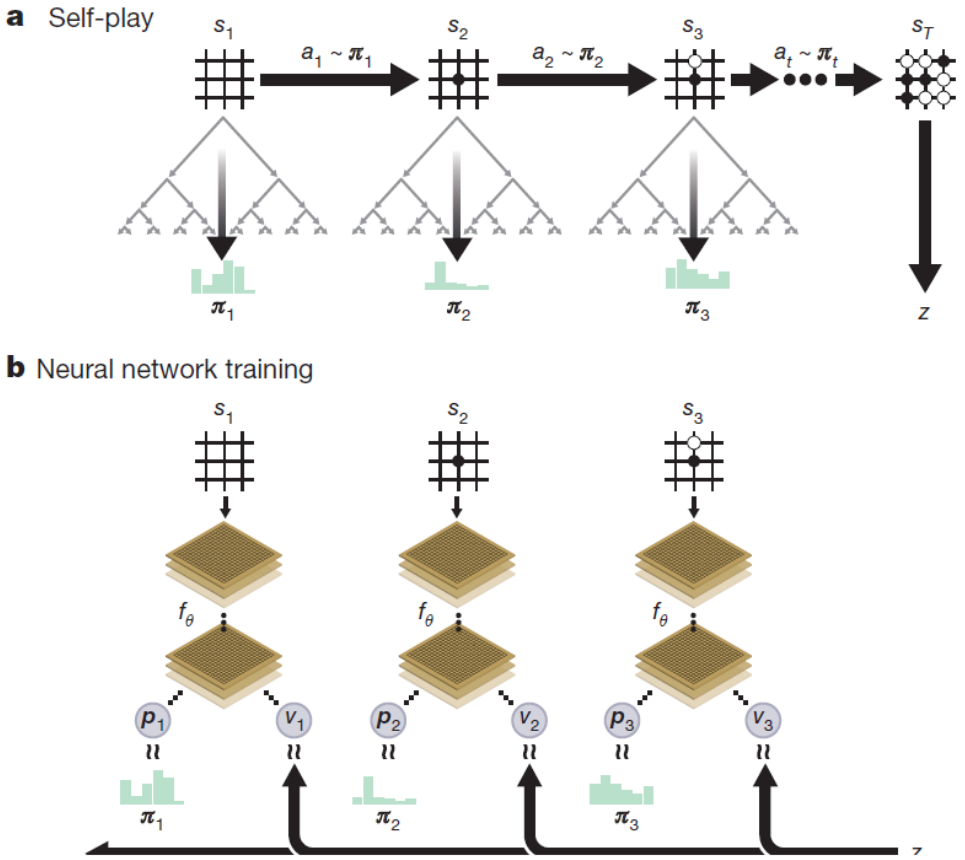
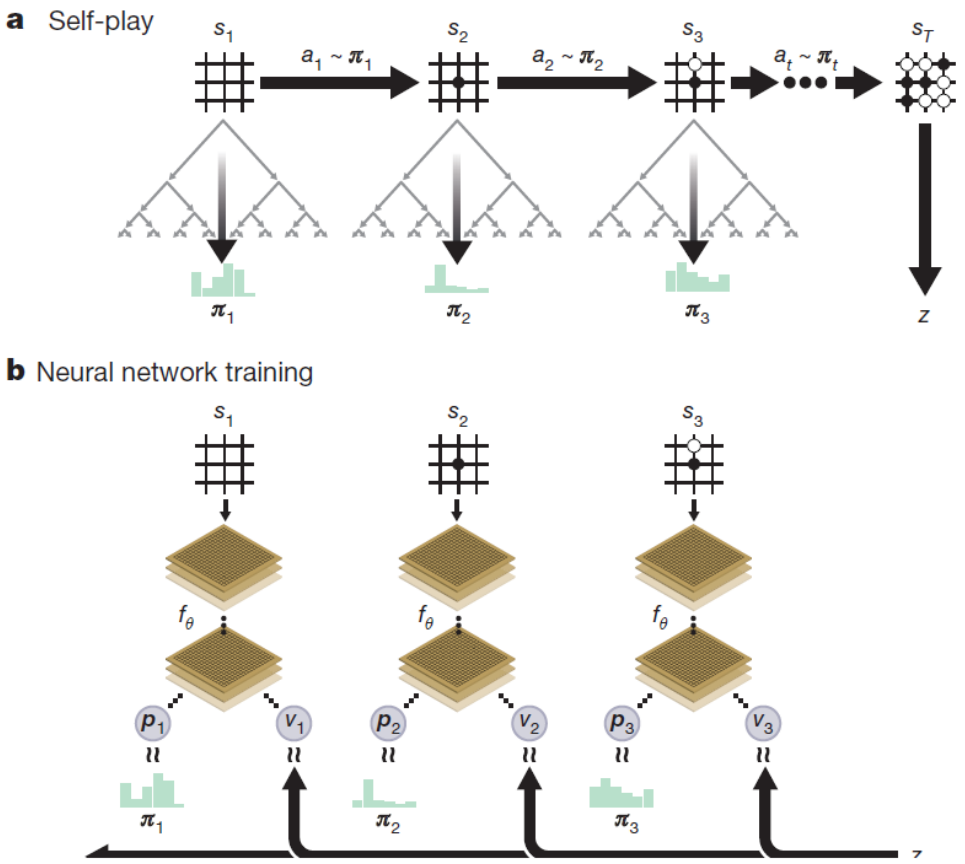


Figure 1 | Self-play reinforcement learning in AlphaGo Zero. a, The program plays a game $s_1, ..., s_T$ against itself. In each position s_t, an MCTS α_θ is executed (see Fig. 2) using the latest neural network f_θ. Moves are selected according to the search probabilities computed by the MCTS, $a_t \sim \boldsymbol{\pi}_t$. The terminal position s_T is scored according to the rules of the game to compute the game winner z. **b**, Neural network training in AlphaGo Zero. The neural network takes the raw board position s_t as its input, passes it through many convolutional layers with parameters θ, and outputs both a vector $\boldsymbol{p}_t$, representing a probability distribution over moves, and a scalar value v_t, representing the probability of the current player winning in position s_t. The neural network parameters θ are updated to maximize the similarity of the policy vector $\boldsymbol{p}_t$ to the search probabilities $\boldsymbol{\pi}_t$, and to minimize the error between the predicted winner v_t and the game winner z (see equation (1)). The new parameters are used in the next iteration of self-play as in **a**.

自对弈

AlphaGo Zero 原理



深度神经网络f_θ将原始的棋盘位置和棋盘的历史作为输入，输出行动概率和价值，$(\boldsymbol{p}, v) = f_\theta(s)$。其中行动概率向量代表选择每一个行动$a$的概率，$p_a = \Pr(a|s)$。标量$v$是一个标量，估计当前的玩家在当前的位置$s$获胜的概率。

首先神经网络被随机初始化一个权重θ_0。在每一次迭代i中，会产生很多次自我对弈的游戏。在每一步骤t，通过使用前面迭代产生的神经网络$f_{\theta_{i-1}}$来指导执行蒙特卡洛树搜索算法通过概率分布$\boldsymbol{\pi}_t$产生一个新的动作。游戏会在满足一定的条件后终止然后打分，赋予一个奖励值$r_T \in \{-1, 1\}$。每一步的数据被存储为：$\{s_t, \pi_t, z_t\}$。然后通过随机梯度下降的算法更新参数$\boldsymbol{\theta}$，使得神经网络的行动概率$\boldsymbol{p}$更加接近搜索概率$\boldsymbol{\pi}$，并且最小化预测价值v与自我对弈的赢家z之间的误差。

目标函数为$l = (z - v)^2 - \pi^T \log(p) + c||\boldsymbol{\theta}||^2$

自对弈

AlphaGo Zero 原理

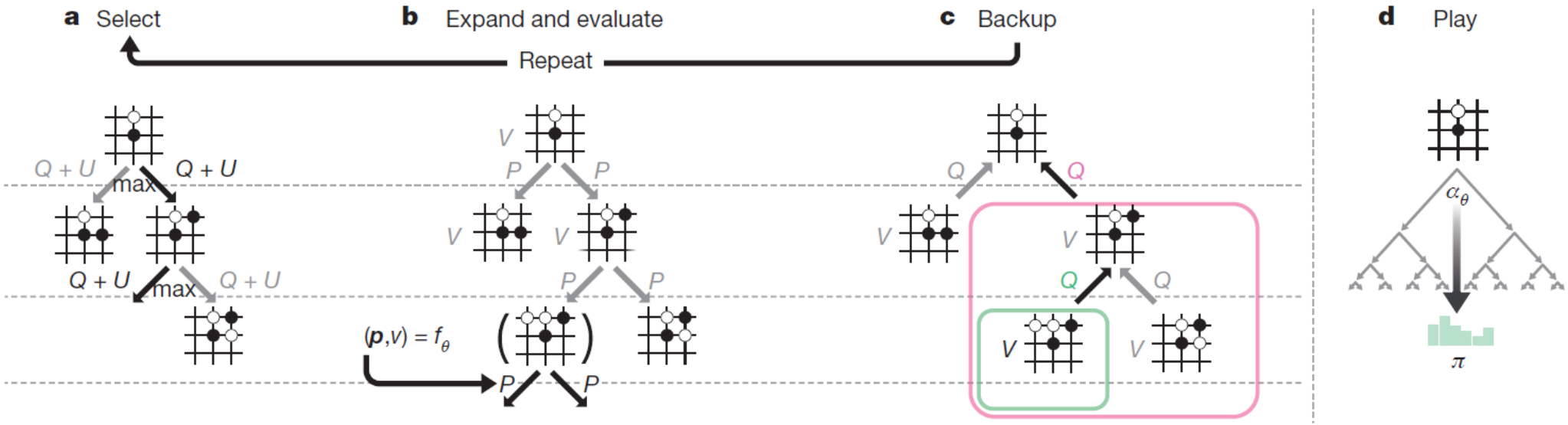
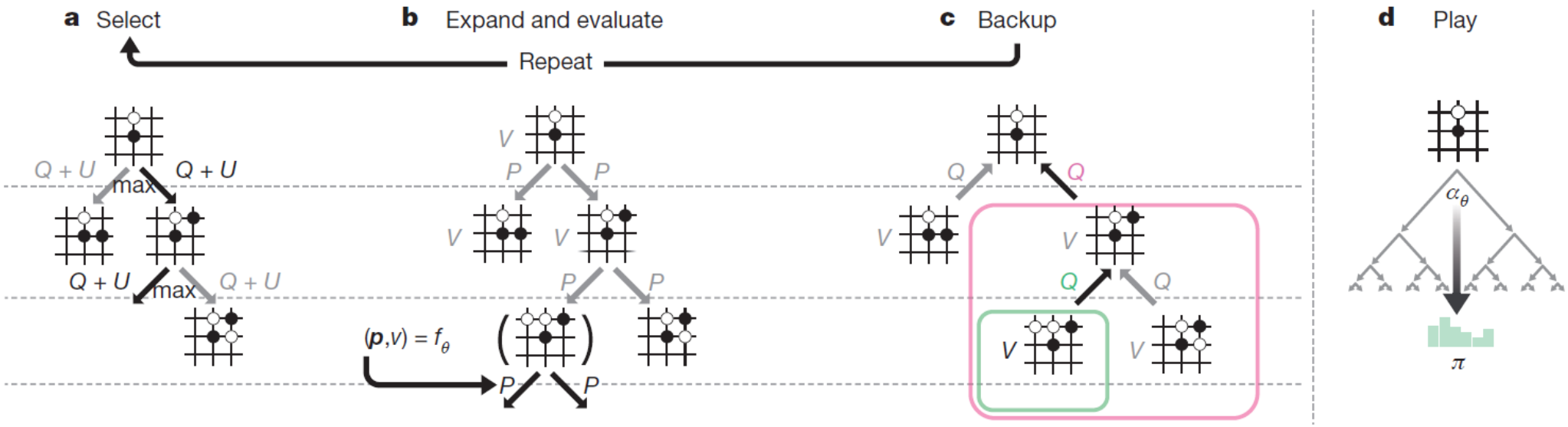


Figure 2 | MCTS in AlphaGo Zero. a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus an upper confidence bound U that depends on a stored prior probability P and visit count N for that edge (which is incremented once traversed). **b**, The leaf node is expanded and the associated position s is evaluated by the neural network $(P(s,\cdot),V(s)) = f_\theta(s)$; the vector of P values are stored in the outgoing edges from s. **c**, Action value Q is updated to track the mean of all evaluations V in the subtree below that action. **d**, Once the search is complete, search probabilities $\boldsymbol{\pi}$ are returned, proportional to $N^{1/\tau}$, where N is the visit count of each move from the root state and τ is a parameter controlling temperature.

MCTS

AlphaGo Zero 原理



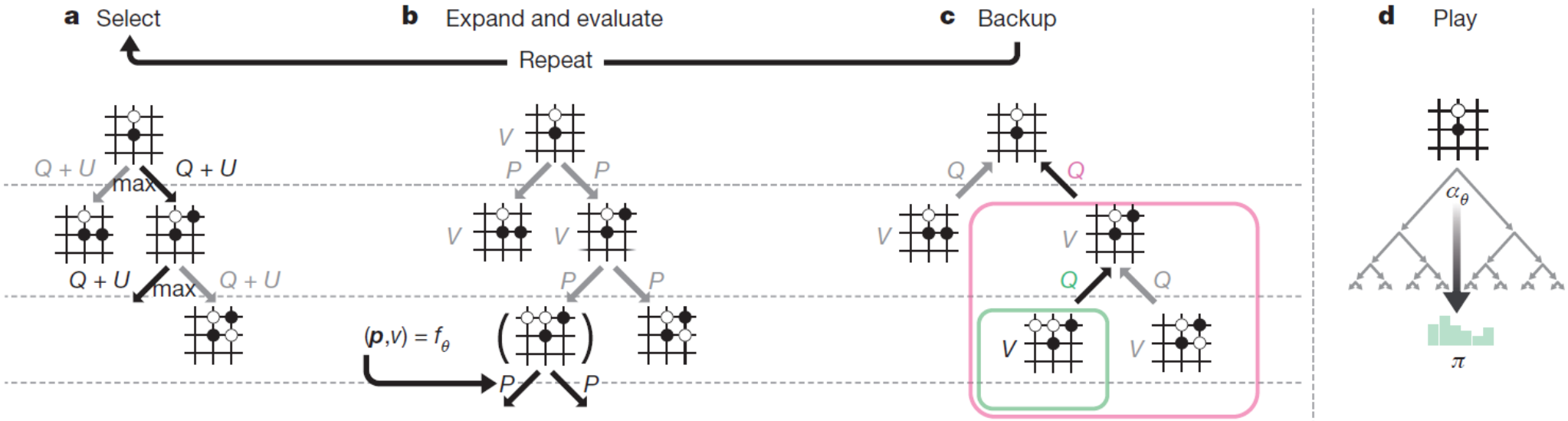
搜索树中的每一个节点s包含了很多的边(s,a)（a为行动空间中所有的合法的动作:$a \in \mathcal{A}(s)$）.每一个边存储了下面的数据

$$\{N(s,a), W(s,a), Q(s,a), P(s,a)\}$$

$N(s,a)$是访问边的次数，$W(s,a)$是总的行动价值，$Q(s,a)$平均行为价值，$P(s,a)$为选择那条边的先验概率。搜索算法分为下面的四步:

MCTS

AlphaGo Zero 原理



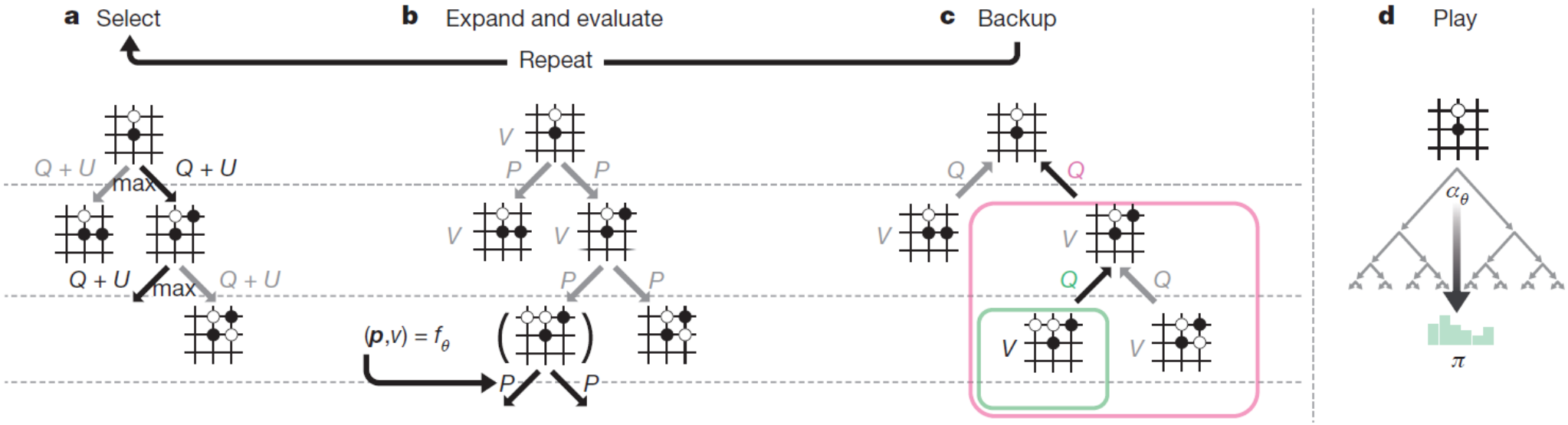
1. **选择** 每一次模拟的树内搜索开始于搜索树的根节点s_0，当模拟在第L步遇到叶子节点s_L的时候，此次模拟结束。当$t < L$时，根据搜索树中的统计数据选择每一个行动，$a_t = \underset{a}{\text{argmax}}(Q(s_t, a) + U(s_t, a))$

$$U(s,a) = c_{puct} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,b)}$$

c_{puct}是一个常量，决定了搜索的广度；搜索控制策略更偏向先验概率更高并且访问次数最低的行动。

MCTS

AlphaGo Zero 原理



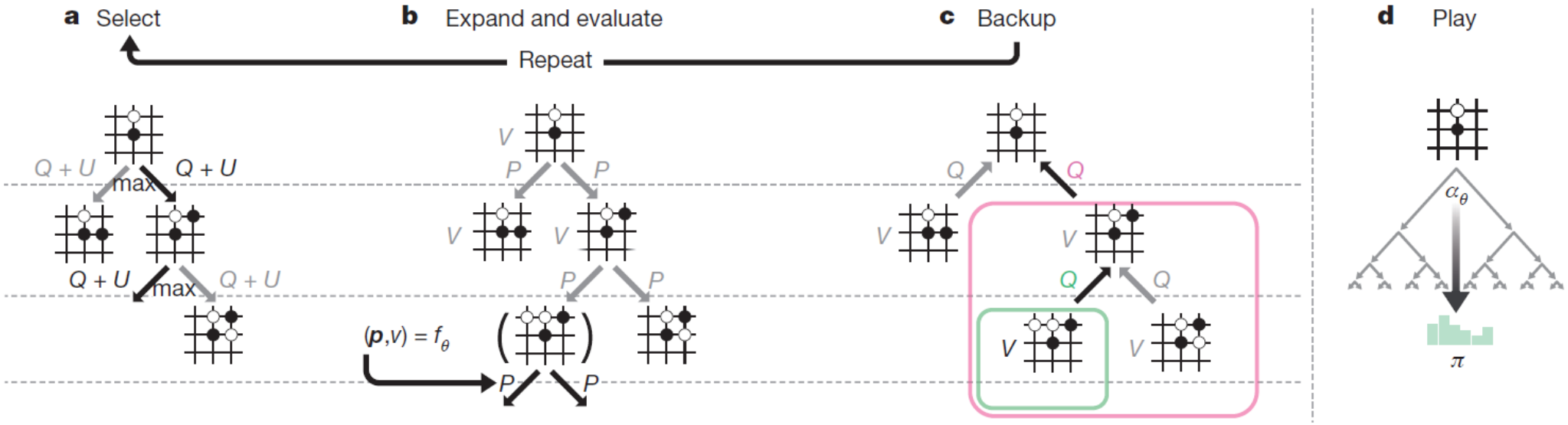
2. **扩展和评估** 叶子被加入到队列中用于神经网络的评估，$(d_i(\boldsymbol{p}), v) = f_\theta(d_i(s_L))$。之后对队列中的位置（Positions）
进行评估。之后叶子节点被扩展，并且每一个边被初始化为下面的值

$$\{N(s_L, a) = 0, W(s_L, a) = 0, Q(s_L, a) = 0, P(s_L, a) = p_a\}$$

评估的价值v会被回溯。

MCTS

AlphaGo Zero 原理

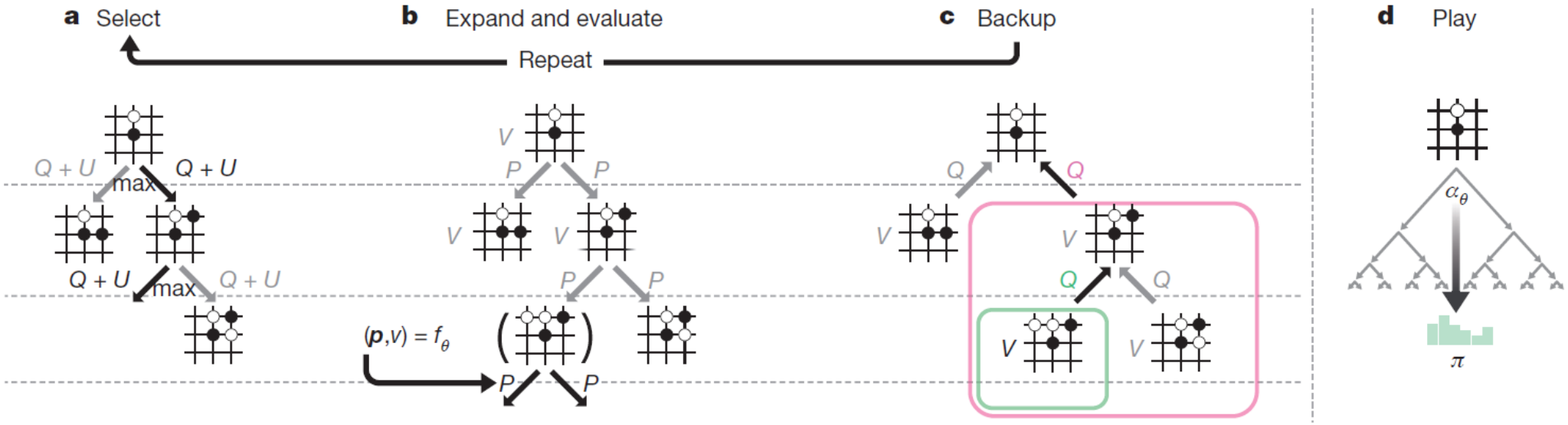


3. **回溯** 通过每一步$t \leq L$的回溯，更新每一个边存储的统计数据：

$$N(s_t, a_t) = N(s_t, a_t) + 1$$
$$W(s_t, a_t) = W(s_t, a_t) + v$$
$$Q(s_t, a_t) = \frac{W(s_t, a_t)}{N(s_t, a_t)}$$

MCTS

AlphaGo Zero 原理



4. Play 在搜索结束之后，Alpha Go Zero会根据返回的搜索概率$\boldsymbol{\pi}(a|s_0) = \frac{N(s_0,a)^{\frac{1}{\tau}}}{\sum_b N(s_0,b)^{\frac{1}{\tau}}}$选择在根位置$s_0$的行动，其中τ是控制搜索广度的温度参数。在后面的步骤中，搜索树根据下面的原则被重新使用：采取上面的行动之后到达的子节点变为新的根节点，这一个子节点以下的子树以及它的全部统计数据被保留，搜索树的其他部分被丢弃。

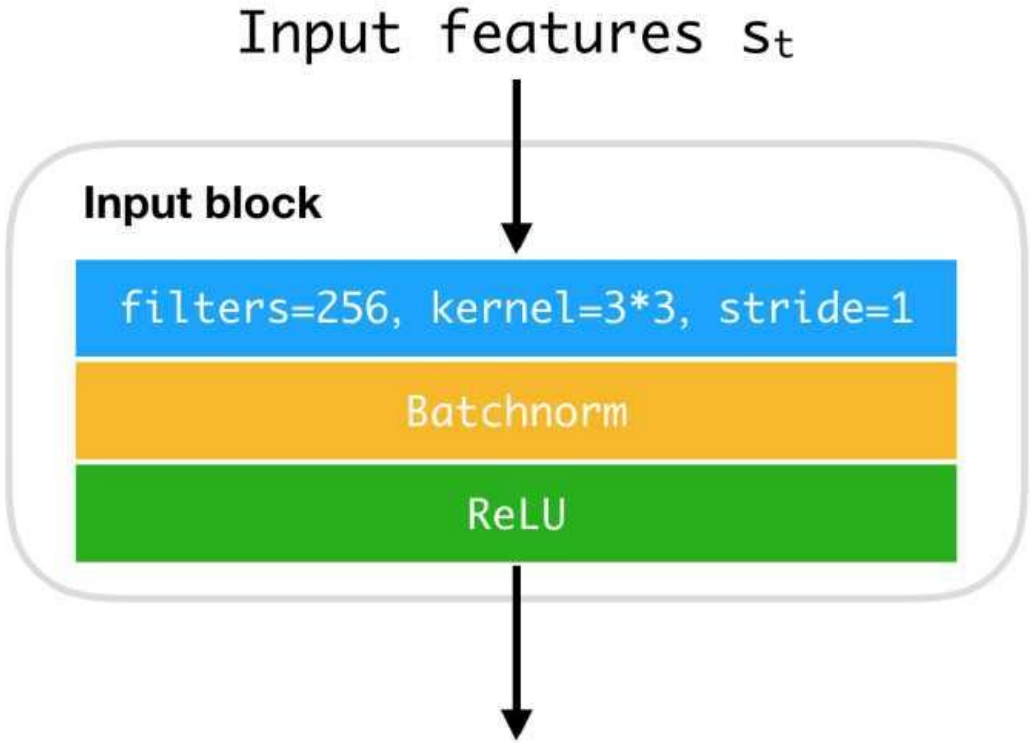
MCTS

AlphaGo

AlphaGo Zero 原理

输入层：

- 采用1的步长，核大小为3 × 3的256个滤波器的卷积；
- 批量规范化；
- 整流非线性化(ReLU)

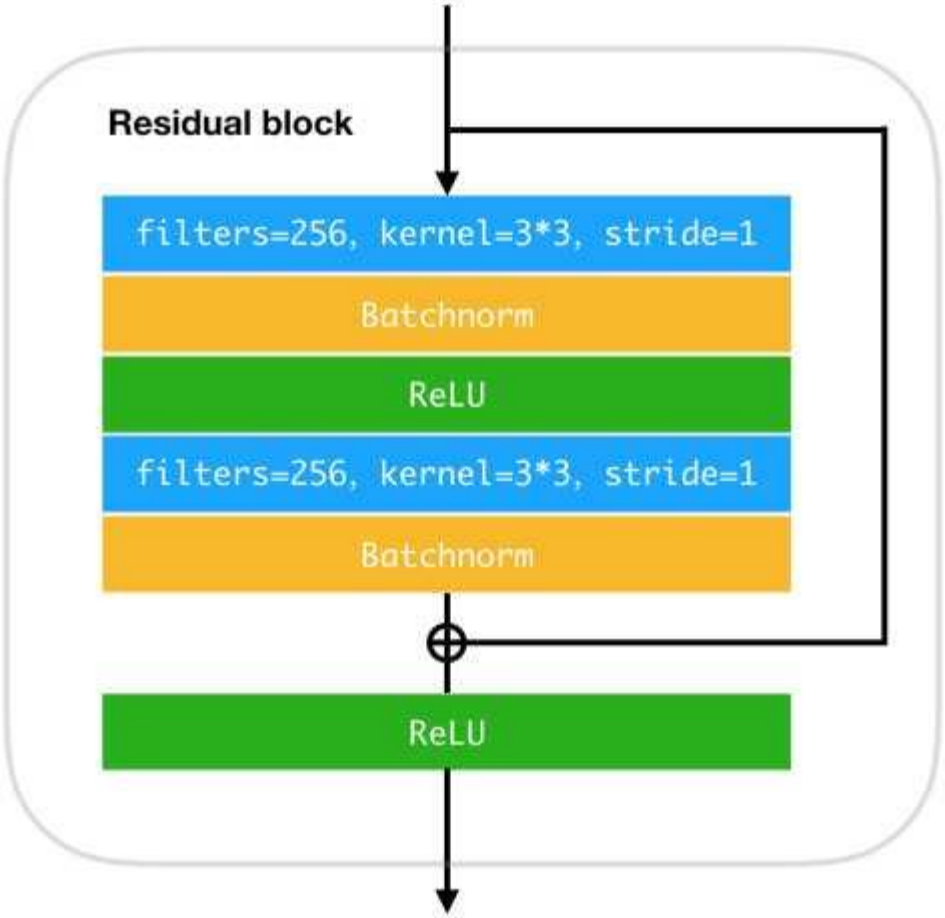


网络结构

AlphaGo Zero 原理

残差模块：

- 采用1的步长，核大小为3 × 3的256个滤波器的卷积；
- 批量规范化；
- 整流非线性化(ReLU)
- 采用1的步长，核大小为3 × 3的256个滤波器的卷积；
- 批量规范化；
- 将输入跳过上面的几步直接作为ReLU的输入
- 整流非线性化(ReLU)



网 络 结 构

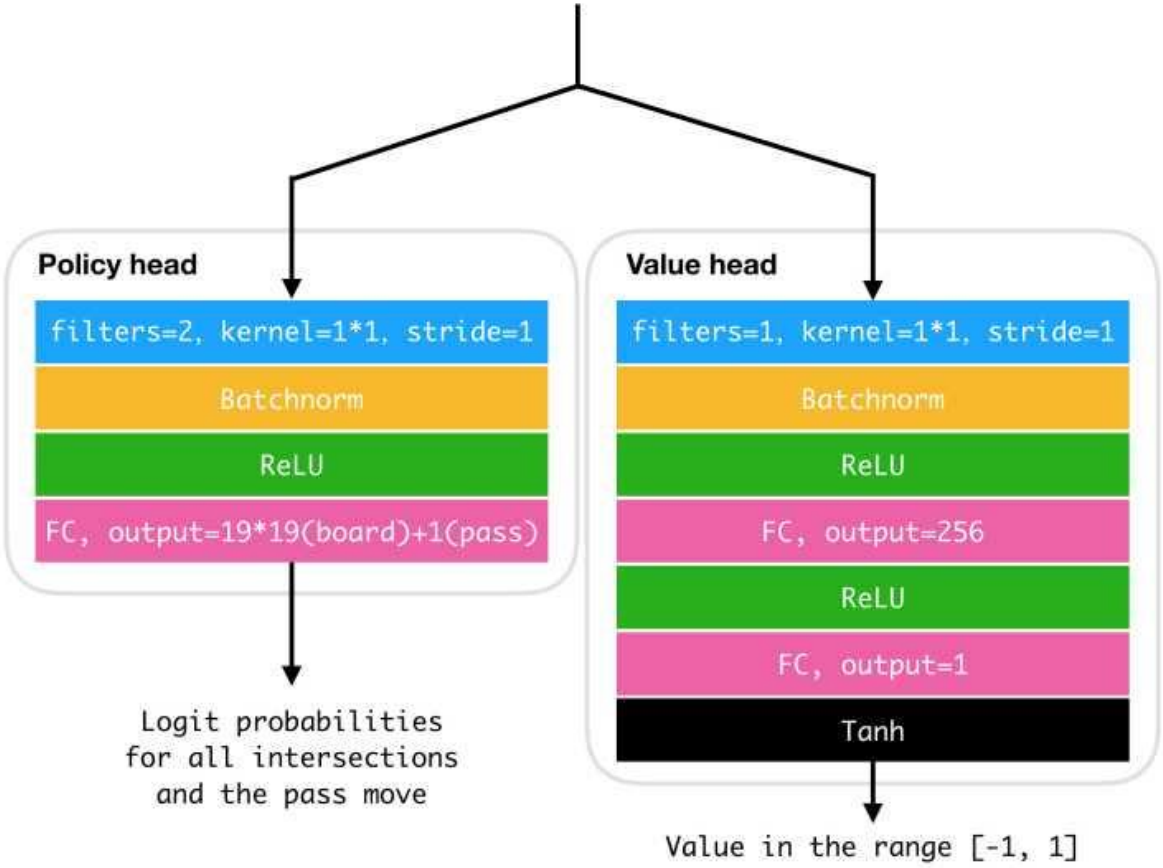
AlphaGo Zero 原理

结尾的策略网络

- 步长1，核大小为1×1的2个滤波器的卷积;
- 批量规范化;
- 整流非线性化(ReLU);
- 输出为 $19^2 + 1 = 362$ 维向量的全连接线性层

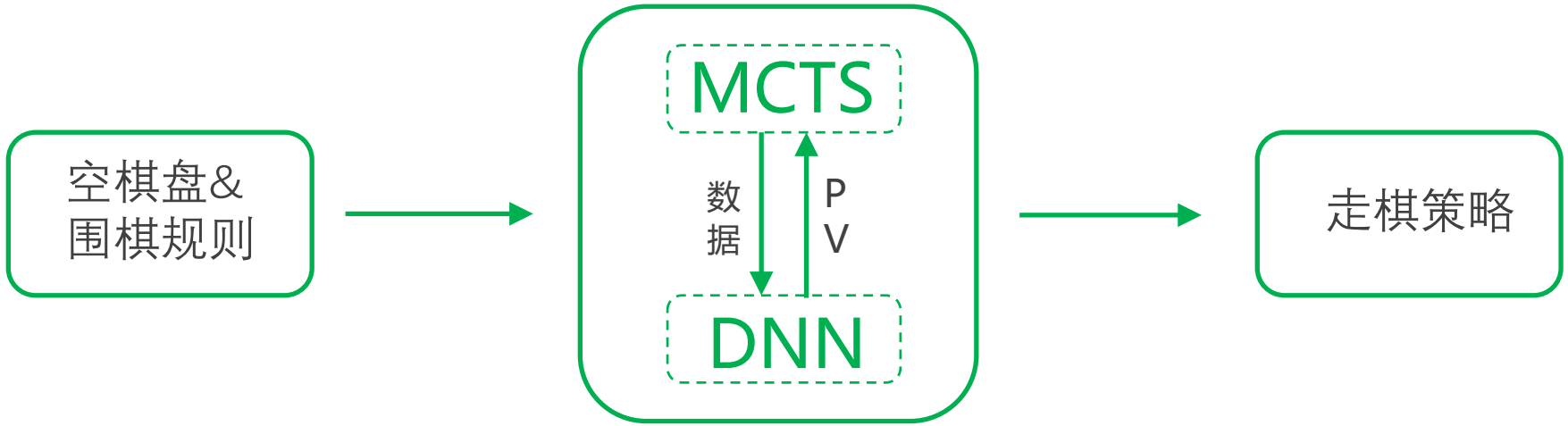
结尾的价值网络

- 步长1，核大小为1×1的1个滤波器的卷积;
- 批量规范化;
- 整流非线性化(ReLU);
- 连接到一个大小为256的隐层的全连接线性层;
- 整流非线性化(ReLU);
- 连接到一个变量的全连接线·性层;
- 输出在[-1,1]范围的tanh非线性单元。



网络结构

AlphaGo Zero 原理



AlphaGo Zero 改进总结

第一，神经网络权值完全随机初始化。不利用任何人类专家的经验或数据，神经网络的权值完全从随机初始化开始，进行随机策略选择，使用强化学习进行自我博弈和提升。

第二，无需先验知识。不再需要人为手工设计特征，而是仅利用棋盘上的黑白棋子的摆放情况，作为原始输入数据，将其输入到神经网络中，以此得到结果。

第三，神经网络结构复杂性降低。原先两个结构独立的策略网络和价值网络合为一体，合并成一个神经网络。在该神经网络中，从输入层到中间层是完全共享的，到最后的输出层部分被分离成了策略函数输出和价值函数输出。

第四，舍弃快速走子网络。不再使用快速走子网络进行随机模拟，而是完全将神经网络得到的结果替换随机模拟，从而在提升学习速率的同时，增强了神经网络估值的准确性。

第五，神经网络引入残差结构。神经网络采用基于残差网络结构的模块进行搭建，用了更深的神经网络进行特征表征提取。从而能在更加复杂的棋盘局面中进行学习。

第六，硬件资源需求更少。以前ELO最高的AlphaGo需要1920块CPU和280块GPU训练，AlphaGoLee则用了176块GPU和48块TPU，而现在，AlphaGoZero则使用了单机4块TPU便能完成训练任务。

第七，学习时间更短。AlphaGoZero仅用3天的时间便能达到AlphaGoLee的水平，21天后达到AlphaGoMaster的水平，棋力提升非常快。

参考资料

1. http://blog.csdn.net/zz_1215/article/details/44138823
2. http://blog.csdn.net/zz_1215/article/details/44138715
3. http://blog.csdn.net/zz_1215/article/details/44138843
4. http://blog.csdn.net/zz_1215/article/details/44138881
5. http://blog.csdn.net/saltriver/article/details/52194918
6. https://www.zhihu.com/question/41176911
7. https://www.zhihu.com/question/20254139
8. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
9. http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html
10. http://www.360doc.com/content/13/0601/00/8076359_289597587.shtml
11. 《机器学习》周志华
12. 《Mastering the Game of Go without Human Knowledge》
13. 《Mastering the Game of Go with deep neural networks and tree search》
14. https://mp.weixin.qq.com/s?__biz=MzIxNjE3MTM5OA==&mid=402241411&idx=1&sn=98557fdc359a17af9ab6b1ed7e09854a&scene=2&srcid=0314rM6ivyxIaEMfKIaW167Z&from=timeline&isappinstalled=0##
15. https://mp.weixin.qq.com/s?__biz=MzI3MTA0MTk1MA==&mid=2652006502&idx=2&sn=11e158df417a3d430183891556736c66&chksm=f1211c97c656958159f36403c631233d20e99008895509ae32f3c064afdf9e350b5f95f6acc4&mpshare=1&scene=1&srcid=1021tgFvRUu4GZ5nCcpv59HH#

完，谢谢！

刘畅 高伟

20171205

AlphaGo